Verifiable and Delegatable Constrained Pseudorandom Functions for Unconstrained Inputs

Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay

Department of Mathematics
Indian Institute of Technology Kharagpur
Kharagpur-721302, India
{pratishdatta,ratna,sourav}@maths.iitkgp.ernet.in

Abstract. *Constrained pseudorandom functions* (CPRF) are a fundamental extension of the notion of traditional pseudorandom functions (PRF). A CPRF enables a master PRF key holder to issue constrained keys corresponding to specific constraint predicates over the input domain. A constrained key can be used to evaluate the PRF only on those inputs which are accepted by the associated constraint predicate. However, the PRF outputs on the rest of the inputs still remain computationally indistinguishable from uniformly random values. A *constrained verifiable pseudorandom function* (CVPRF) enhances a CPRF with a non-interactive public verification mechanism for checking the correctness of PRF evaluations. A *delegatable constrained pseudorandom function* (DCPRF) is another extension which augments a CPRF to empower constrained key holders to delegate further constrained keys that allow PRF evaluations on inputs accepted by more restricted constraint predicates compared to ones embedded in their own constrained keys. Until recently, all the proposed constructions of CPRF's and their extensions(i) either could handle only bounded length inputs, (ii) or were based on risky knowledge-type assumptions. In EUROCRYPT 2016, Deshpande et al. have presented a CPRF construction supporting inputs of *unconstrained* polynomial length based on indistinguishability obfuscation and injective pseudorandom generators, which they have claimed to be selectively secure. In this paper, we first identify a flaw in their security argument and resolve this by carefully modifying their construction and suitably redesigning the security proof. Our alteration does not involve any additional heavy duty cryptographic tools. Next, employing only standard public key encryption (PKE), we extend our CPRF construction, presenting the *first* ever CVPRF and DCPRF constructions that can handle inputs of *unbounded* polynomial length. Finally, we apply our ideas to demonstrate the *first* known *attribute-based signature* (ABS) scheme for *general* signing policies supporting signing attributes of *arbitrary* polynomial length.

Keywords: constrained pseudorandom functions, verifiable constrained pseudorandom function, key delegation, indistinguishability obfuscation

1 Introduction

Constrained Pseudorandom Functions: *Constrained pseudorandom functions* (CPRF), concurrently introduced by Boneh and Waters [BW13], Boyle et al. [BGI14], as well as Kiayias et al. [KPTZ13], are promising extension of the notion of standard *pseudorandom functions* (PRF) [GGM86] – a fundamental primitive in modern cryptography. A standard PRF is a deterministic keyed function with the following property: Given a key, the function can be computed in polynomial time at all points of its input domain. But, without the key it is computationally hard to distinguish the PRF output at any arbitrary input from a uniformly random value, even after seeing the PRF evaluations on a polynomial number of inputs. A CPRF is an augmentation of a PRF with an additional *constrain algorithm* which enables a party holding a PRF key, also referred to as a master PRF key, to derive constrained keys that allow the evaluation of the PRF over certain subsets of the input domain. However, PRF evaluations on the rest of the inputs still remain computationally indistinguishable from random.

Since their inception, CPRF's have found countless interesting applications in various branches of cryptography ranging from broadcast encryption [BW13] and attribute-based encryption [DKW16] to policy-based key distribution [BW13] and multi-party (identity-based) non-interactive key exchange [BZ14] ([BW13]). Even the simplest class of CPRF's, known as *puncturable pseudorandom functions* (PPRF) [SW14], have turned out to be a powerful tool in conjunction

with indistinguishability obfuscation [GGH$^+$13, GLSW15]. In fact, the combination of these two primitives have led to solutions of longstanding open problems including deniable encryption [SW14], full domain hash [HSW14], adaptively secure functional encryption for general functionalities [Wat15], and functional encryption for randomized functionalities [GJKS15] through the classic punctured programming technique introduced in [SW14].

Over the last few years there has been a significant progress in the field of CPRF's. In terms of expressiveness of the constraint predicates, starting with the most basic type of constraints such as prefix constraints [BW13, BGI14, KPTZ13] (which also encompass puncturing constraints) and bit fixing constraints [BW13, FKPR14], CPRF's have been constructed for highly rich constraint families such as circuit constraints [BW13, BV15, BFP$^+$15, HKKW14] employing diverse cryptographic tools and based on various complexity assumptions. In terms of security, most of the existing CPRF constructions are only *selectively* secure. The stronger and more realistic notion of *adaptive* security seems to be rather challenging to achieve without complexity leveraging. In fact, the best known results so far on adaptive security of CPRF's require super-polynomial security loss [FKPR14], or work for very restricted form of constraints [HKW15], or attain the security in non-collusion mode [BV15], or accomplish security in the random oracle model [HKKW14].

Constrained Verifiable Pseudorandom Functions: An interesting enhancement of the usual CPRF's is *verifiability*. A *verifiable constrained pseudorandom function* (CVPRF), independently introduced by [Fuc14, CRV14], is the unification of the notions of a *verifiable random function* (VRF) [MRV99] and a standard CPRF. In a CVPRF system, just like a traditional VRF, a public verification key is set along with the master PRF key. Besides enabling the evaluation of the PRF, the master PRF key can be utilized to generate a non-interactive proof of the correctness of evaluation. This proof can be verified by any party using only the public verification key. On the other hand, as in the case of a CPRF, here also the master PRF key holder can give out constrained keys for specific constraint predicates. A constrained key corresponding to some constraint predicate p allows the evaluation of the PRF together with the generation of a non-interactive proof of correct evaluation for only those inputs x for which $p(x) = 1$. In essence, CVPRF's resolve the issue of trust on a CPRF evaluator for the correctness of the received PRF output. In [Fuc14, CRV14], the authors have shown that the CPRF constructions of [BW13] for the bit fixing and circuit constraints can be augmented with the verifiability feature without incurring any significant additional cost.

Delegatable Constrained Pseudorandom Functions: *Key delegation* is another vital enrichment of standard CPRF's. This feature empowers the holder of a constrained key, corresponding to some constraint predicate p belonging to certain constraint family $\mathbb{P}$ over the input domain of the PRF, with the ability to distribute further restricted keys corresponding to the joint predicates $p \wedge \widetilde{p}$, for constraints $\widetilde{p} \in \mathbb{P}$. Such a delegated key can be utilized to evaluate the PRF on only those inputs x for which $[p(x) = 1] \wedge [\widetilde{p}(x) = 1]$, whereas, the PRF outputs on the rest of the inputs are computationally indistinguishable from random values. The concept of key delegation in the context of CPRF's has been recently introduced by [CRV14], who have also shown how to extend the bit fixing and circuit-based CPRF constructions of [BW13] to support key delegation.

CPRF's for Unconstrained Inputs: Until recently, the research on CPRF's has been confined to inputs of apriori bounded length. In fact, all the CPRF constructions mentioned above could handle only bounded length inputs. Abusalah et al. [AFP14] have taken a first step forward towards overcoming the barrier of bounded input length. They have also demonstrated highly motivating applications of CPRF's supporting apriori unbounded or unconstrained length inputs such as broadcast encryption with an unbounded number of recipients and multi-party

identity-based non-interactive key exchange with no pre-determined bound on the number of parties. They presented a selectively secure CPRF for unconstrained length inputs by viewing the constraint predicates as *Turing machines* (TM) that can handle inputs of arbitrary polynomial length. In a more recent work, Abusalah and Fuchsbauer [AF16] have made progress towards efficiency improvements by constructing TM-based CPRF's with much shorter constrained keys compared to the CPRF construction of [AFP14].

However, both the aforementioned CPRF constructions rely on the existence of public-coin differing-input obfuscators [IPS15] and succinct non-interactive arguments of knowledge [BCCT12, GW11], which are believed to be risky assumptions due to their inherent extractability nature. Deshpande et al. [DKW16] have very recently built a selectively secure CPRF for TM constraints, supporting inputs of unbounded polynomial length, based on the existence of indistinguishability obfuscators (IO) for circuits and injective pseudorandom generators. Currently, there is no known impossibility or implausability result on IO and, moreover, in the last few years, there has been a significant progress towards constructing IO based on standard complexity assumptions [GLSW15, AJ15, AJS15]. Unfortunately however, as we find out in this paper, the security argument of [DKW16] has a flaw. In view of this fact, the ambitious goal of constructing CPRF's for *unconstrained* inputs from *IO for circuits* still remains unaddressed. Furthermore, all known constructions of CVPRF's and CPRF's endored with key delegation can only support inputs of apriori bounded length.

Our Contributions

Our work in this paper is manifold:

- We first resolve the flaw in the security argument of the CPRF construction of [DKW16] by carefully modifying the construction and suitably redesigning the security proof. While modifying the CPRF construction of [DKW16], we only additionally use a somewhere statistically binding (SSB) hash function [HW15, OPWW15] beyond the cryptographic tools used by [DKW16]. In effect, our modified CPRF construction stands out as the *first* IO-based selectively secure CPRF system for TM constraints that can handle inputs of arbitrary polynomial length.

- Our second and more significant contribution is to enhance our CPRF construction with verifiability and key delegation features, thereby, developing the *first* IO-based selectively secure CVPRF and delegatable CPRF (DCPRF) constructions supporting inputs of *unbounded* polynomial length. Towards achieveing those two augmentations of our CPRF, we only assume the existence of a perfectly correct and chosen plaintext attack (CPA) secure public key encryption scheme, which is evidently a minimal assumption.

- Finally, applying an analogous idea as the one used in our CVPRF construction, we build the *first* (key-policy) *attribute-based signature* (ABS) scheme for *general* signing policies supporting signing attributes of *arbitrary* polynomial length. In a (key-policy) ABS scheme [MPR11, OT14, TLL14, SAH16], a setup authority holds a master signing key and publishes system public parameters. Using its master signing key, the authority can distribute restricted signing keys corresponding to specific signing policies. Such a constrained signing key enables a signer to sign messages with respect to only those signing attributes which are accepted by the signing policy embedded within the signing key. The signatures are verifiable by anyone using solely the public parameters. By verifying a signature on some message with respect to some signing attributes, a verifier gets convinced that the signature is indeed generated by a signer possessing a signing key corresponding to some signing policy that accepts the signing attributes. However, the verifier cannot trace the exact signer or signing policy used to generate the signature. Our ABS only uses an existentially unforgeable digital signature scheme in

addition to the cryptographic building blocks employed in our CPRF construction. We note that using a similar technique as that of our DCPRF construction, our ABS scheme can be further enriched to support delegation of signing keys. We would also like to mention that using the technique of universal TM, our key-policy ABS construction can be transformed into a ciphertext-policy variant.

2 Preliminaries

Here we give the necessary background on various cryptographic primitives we will be using throughout this paper. Let $\lambda \in \mathbb{N}$ denotes the security parameter. For $n \in \mathbb{N}$ and $a, b \in \mathbb{N} \cup \{0\}$ (with $a < b$), we let $[n] = \{1, \ldots, n\}$ and $[a, b] = \{a, \ldots, b\}$. For any set S, $\upsilon \xleftarrow{\$} S$ represents the uniform random variable on S. For a randomized algorithm $\mathcal{R}$, we denote by $\psi = \mathcal{R}(\upsilon; \rho)$ the random variable defined by the output of $\mathcal{R}$ on input υ and randomness ρ, while $\psi \xleftarrow{\$} \mathcal{R}(\upsilon)$ has the same meaning with the randomness suppressed. Also, if $\mathcal{R}$ is a deterministic algorithm $\psi = \mathcal{R}(\upsilon)$ denotes the output of $\mathcal{R}$ on input υ. We will use the alternative notation $\mathcal{R}(\upsilon) \to \psi$ as well to represent the output of the algorithm $\mathcal{R}$, whether randomized or deterministic, on input υ. For any string $s \in \{0,1\}^*$, $|s|$ represents the length of the string s. For any two strings $s, s' \in \{0,1\}^*$, $s \| s'$ represents the concatenation of s and s'. A function negl is *negligible* if for every integer c, there exists an integer k such that for all $\lambda > k$, $|\mathsf{negl}(\lambda)| < 1/\lambda^c$.

2.1 Turing Machines

A Turing machine (TM) M is a 7-tuple $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$ with the following semantics:

- Q: The finite set of possible states of M.
- Σ_{INP}: The finite set of input symbols.
- Σ_{TAPE}: The finite set of tape symbols such that $\Sigma_{\text{INP}} \subset \Sigma_{\text{TAPE}}$ and there exists a special blank symbol '_' $\in \Sigma_{\text{TAPE}} \backslash \Sigma_{\text{INP}}$.
- $\delta : Q \times \Sigma_{\text{TAPE}} \to Q \times \Sigma_{\text{TAPE}} \times \{+1, -1\}$: The transition function of M.
- $q_0 \in Q$: The designated start state.
- $q_{\text{AC}} \in Q$: The designated accept state.
- $q_{\text{REJ}} (\neq q_{\text{AC}}) \in Q$: The distinguished reject state.

For any $t \in [T = 2^\lambda]$, we define the following variables for M, while running on some input (without the explicit mention of the input in the notations):

- $\text{POS}_{M,t}$: An integer which denotes the position of the header of M after the t^{th} step. Initially, $\text{POS}_{M,0} = 0$.
- $\text{SYM}_{M,t} \in \Sigma_{\text{TAPE}}$: The symbol stored on the tape at the $\text{POS}_{M,t}{}^{\text{th}}$ location.
- $\text{SYM}_{M,t}^{(\text{WRITE})} \in \Sigma_{\text{TAPE}}$: The symbol to be written at the $\text{POS}_{M,t-1}{}^{\text{th}}$ location during the t^{th} step.
- $\text{ST}_{M,t} \in Q$: The state of M after the t^{th} step. Initially, $\text{ST}_{M,0} = q_0$.

At each time step, theTM M reads the tape at the header position and based on the current state, computes what needs to be written on the tape at the current header location, the next state, and whether the header must move left or right. More formally, let $(q, \zeta, \beta \in \{+1, -1\}) = \delta(\text{ST}_{M,t-1}, \text{SYM}_{M,t-1})$. Then, $\text{ST}_{M,t} = q, \text{SYM}_{M,t}^{(\text{WRITE})} = \zeta$, and $\text{POS}_{M,t} = \text{POS}_{M,t-1} + \beta$. M accepts at time t if $\text{ST}_{M,t} = q_{\text{AC}}$. In this paper we consider $\Sigma_{\text{INP}} = \{0,1\}$ and $\Sigma_{\text{TAPE}} = \{0,1,\_\}$. Given any TM M and string $x \in \{0,1\}^*$, we define $M(x) = 1$, if M accepts x within T steps, and 0, otherwise.

2.2 Indistinguishability Obfuscation

The following formalization of indistinguishability obfuscation (IO) is due to Garg et al. [GGH+13].

Definition 2.1 (Indistinguishability Obfuscation: IO). An indistinguishability obfuscator (IO) $\mathcal{IO}$ for a circuit class $\{\mathbb{C}_\lambda\}_\lambda$ is a probabilistic polynomial-time (PPT) uniform algorithm satisfying the following conditions:

- **Correctness**: $\mathcal{IO}(1^\lambda, C)$ preserves the functionality of the input circuit C, i.e., for any $C \in \mathbb{C}_\lambda$, if we compute $C' = \mathcal{IO}(1^\lambda, C)$, then $C'(v) = C(v)$ for all inputs v.

- **Indistinguishability**: For any security parameter λ and any two circuits $C_0, C_1 \in \mathbb{C}_\lambda$ with the same functionality, the circuits $\mathcal{IO}(1^\lambda, C_0)$ and $\mathcal{IO}(1^\lambda, C_1)$ are computationally indistinguishable. More precisely, for all (not necessarily uniform) PPT adversaries $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$, there exists a negligible function negl such that, if

$$\mathsf{Pr}\big[(C_0, C_1, \xi) \xleftarrow{\$} \mathcal{D}_1(1^\lambda) \ : \ \forall\, v, C_0(v) = C_1(v)\big] \geq 1 - \mathsf{negl}(\lambda),$$
$$\text{then } \big|\mathsf{Pr}\big[\mathcal{D}_2(\xi, \mathcal{IO}(1^\lambda, C_0)) = 1\big] - \mathsf{Pr}\big[\mathcal{D}_2(\xi, \mathcal{IO}(1^\lambda, C_1)) = 1\big]\big| \leq \mathsf{negl}(\lambda).$$

We remark that the two distinct algorithms $\mathcal{D}_1$ and $\mathcal{D}_2$, which pass state ξ, can be viewed equivalently as a single stateful algorithm $\mathcal{D}$. In this paper we employ the latter approach, although here we present the definition as it appears in [GGH+13]. When clear from the context, we will drop 1^λ as an input to $\mathcal{IO}$ and λ as a subscript of $\mathbb{C}$.

The circuit class we are interested in are polynomial-size circuits, i.e., when $\mathbb{C}_\lambda$ is the collection of all circuits of size at most λ. This circuit class is denoted by P/poly. The first candidate construction of IO for P/poly was presented by Garg et al. [GGH+13] in a generic model of encoded matrices. Later, Pass et al. [PST14] and Gentry et al. [GLSW15] have shown that IO for P/poly can be developed based on a single instance-independent assumption.

2.3 Puncturable Pseudorandom Function

Puncturable pseudorandom functions, first defined by Sahai and Waters [SW14], are a special class of constrained pseudorandom functions, which we will formally define in Section 3.1. Here we present the definition of puncturable pseudorandom functions following [SW14].

Definition 2.2 (Puncturable Pseudorandom Function: PPRF). A puncturable pseudorandom function (PPRF) $\mathcal{F} : \mathcal{K}_{\text{PPRF}} \times \mathcal{X}_{\text{PPRF}} \to \mathcal{Y}_{\text{PPRF}}$ consists of an additional punctured key space $\mathcal{K}_{\text{PPRF-PUNC}}$ other than the usual key space $\mathcal{K}_{\text{PPRF}}$ and PPT algorithms ($\mathcal{F}$.Setup, $\mathcal{F}$.Eval, $\mathcal{F}$.Puncture, $\mathcal{F}$.Eval-Punctured) described below. Here, $\mathcal{X}_{\text{PPRF}} = \{0,1\}^{\ell_{\text{PPRF-INP}}}$ and $\mathcal{Y}_{\text{PPRF}} = \{0,1\}^{\ell_{\text{PPRF-OUT}}}$, where $\ell_{\text{PPRF-INP}}$ and $\ell_{\text{PPRF-OUT}}$ are polynomials in the security parameter λ,

$\mathcal{F}.\mathsf{Setup}(1^\lambda) \to K$: The setup authority takes as input the security parameter 1^λ and uniformly samples a PPRF key $K \in \mathcal{K}_{\text{PPRF}}$.

$\mathcal{F}.\mathsf{Eval}(K, x) \to r$: The setup authority takes as input a PPRF key $K \in \mathcal{K}_{\text{PPRF}}$ along with an input $x \in \mathcal{X}_{\text{PPRF}}$. It outputs the PPRF value $r \in \mathcal{Y}_{\text{PPRF}}$ on x. For simplicity, we will represent by $\mathcal{F}(K, x)$ the output of this algorithm.

$\mathcal{F}.\mathsf{Puncture}(K, x) \to K\{x\}$: Taking as input a PPRF key $K \in \mathcal{K}_{\text{PPRF}}$ along with an element $x \in \mathcal{X}_{\text{PPRF}}$, the setup authority outputs a punctured key $K\{x\} \in \mathcal{K}_{\text{PPRF-PUNC}}$.

$\mathcal{F}.\mathsf{Eval\text{-}Puncured}(K\{x\}, x') \rightarrow r$ or $\perp$: An evaluator takes as input a punctured key $K\{x\} \in \mathcal{K}_{\text{PPRF-PUNC}}$ along with an input $x' \in \mathcal{X}_{\text{PPRF}}$. It outputs either a value $r \in \mathcal{Y}_{\text{PPRF}}$ or a distinguished symbol $\perp$ indicating failure. For simplicity, we will represent by $\mathcal{F}(K\{x\}, x')$ the output of this algorithm.

The algorithms $\mathcal{F}.\mathsf{Setup}$ and $\mathcal{F}.\mathsf{Puncture}$ are randomized, whereas, the algorithms $\mathcal{F}.\mathsf{Eval}$ and $\mathcal{F}.\mathsf{Eval\text{-}Punctured}$ are deterministic. The algorithms satisfy the following properties:

► **Correctness under Puncturing**: Consider any security parameter λ, $K \in \mathcal{K}_{\text{PPRF}}$, $x \in \mathcal{X}_{\text{PPRF}}$, and $K\{x\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K, x)$. Then it must hold that

$$\mathcal{F}(K\{x\}, x') = \begin{cases} \mathcal{F}(K, x'), \text{ if } x' \neq x \\ \perp, \qquad \text{otherwise} \end{cases}$$

► **Selective Pseudorandomness at Punctured Points**: This property of a PPRF is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ submits a challenge input $x^* \in \mathcal{X}_{\text{PPRF}}$ to $\mathcal{C}$.
- $\mathcal{C}$ chooses uniformly at random a PPRF key $K^* \xleftarrow{\$} \mathcal{K}_{\text{PPRF}}$ and a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. It computes the punctured key $K^*\{x^*\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^*, x^*)$. If $\hat{b} = 0$, it sets $r^* = \mathcal{F}(K^*, x^*)$. Otherwise, it selects $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$. It sends back $(K^*\{x^*\}, r^*)$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The PPRF $\mathcal{F}$ is selectively pseudorandom at punctured points if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\mathcal{F},\text{SEL-PR}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

Boneh and Waters [BW13], Boyle et al. [BGI14], as well as Kiayias et al. [KPTZ13] have concurrently shown that the tree-based PRF constructed by Goldreich et al. [GGM86] can be modified in a straightforward fashion to build a PPRF from one-way functions.

2.4 IO-Compatible Cryptographic Primitives

In this section, we present certain IO-friendly cryptographic tools which we will be using in the sequel.

2.4.1 Somewhere Statistically Binding Hash Function

We provide the definition of somewhere statistically binding hash function as defined by Hubacek et al. [HW15].

Definition 2.3 (Somewhere Statistically Binding Hash Function: SSB Hash). A somewhere statistically binding (SSB) hash consists of the PPT algorithms $(\mathsf{SSB.Gen}, \mathcal{H}, \mathsf{SSB.Open}, \mathsf{SSB.Verify})$ along with a block alphabet $\Sigma_{\text{SSB-BLK}} = \{0,1\}^{\ell_{\text{SSB-BLK}}}$, output size $\ell_{\text{SSB-HASH}}$, and opening space $\Pi_{\text{SSB}} = \{0,1\}^{\ell_{\text{SSB-OPEN}}}$, where $\ell_{\text{SSB-BLK}}, \ell_{\text{SSB-HASH}}, \ell_{\text{SSB-OPEN}}$ are some polynomials in the security parameter λ. The algorithms have the following syntax:

$\mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}}, i^*) \rightarrow \text{HK}$: The setup authority takes as input the security parameter 1^λ, an integer $n_{\text{SSB-BLK}} \leq 2^\lambda$ representing the maximum number of blocks that can be hashed, and an index $i^* \in [0, n_{\text{SSB-BLK}} - 1]$ and publishes a public hashing key HK.

$\mathcal{H}_{\text{HK}} : x \in \Sigma_{\text{SSB-BLK}}^{n_{\text{SSB-BLK}}} \to h \in \{0,1\}^{\ell_{\text{SSB-HASH}}}$: This is a deterministic function that has the hash key HK hardwired. A user runs this function on input $x = x_0 \| \ldots \| x_{n_{\text{SSB-BLK}}-1} \in \Sigma_{\text{SSB-BLK}}^{n_{\text{SSB-BLK}}}$ to obtain as output $h = \mathcal{H}_{\text{HK}}(x) \in \{0,1\}^{\ell_{\text{SSB-HASH}}}$.

$\mathsf{SSB.Open}(\text{HK}, x, i) \to \pi_{\text{SSB}}$: Taking as input the hash key HK, input $x \in \Sigma_{\text{SSB-BLK}}^{n_{\text{SSB-BLK}}}$, and an index $i \in [0, n_{\text{SSB-BLK}} - 1]$, a user creates an opening $\pi_{\text{SSB}} \in \Pi_{\text{SSB}}$.

$\mathsf{SSB.Verify}(\text{HK}, h, i, u, \pi_{\text{SSB}}) \to \hat{\beta} \in \{0,1\}$: On input a hash key HK, a hash value $h \in \{0,1\}^{\ell_{\text{SSB-HASH}}}$, an index $i \in [0, n_{\text{SSB-BLK}} - 1]$, a value $u \in \Sigma_{\text{SSB-BLK}}$, and an opening $\pi_{\text{SSB}} \in \Pi_{\text{SSB}}$, a verifier outputs a bit $\hat{\beta} \in \{0,1\}$.

The algorithms SSB.Gen and SSB.Open are randomized, while the algorithm SSB.Verify is deterministic. An SSB hash satisfies the following properties:

► **Correctness**: For any security parameter λ, integer $n_{\text{SSB-BLK}} \leq 2^\lambda$, $i, i^* \in [0, n_{\text{SSB-BLK}} - 1]$, $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}}, i^*)$, $x \in \Sigma_{\text{SSB-BLK}}^{n_{\text{SSB-BLK}}}$, and $\pi_{\text{SSB}} \xleftarrow{\$} \mathsf{SSB.Open}(\text{HK}, x, i)$, we have $\mathsf{SSB.Verify}(\text{HK}, \mathcal{H}_{\text{HK}}(x), i, x_i, \pi_{\text{SSB}}) = 1$.

► **Index Hiding**: The index hiding property of an SSB hash is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ chooses an integer $n_{\text{SSB-BLK}} \leq 2^\lambda$ together with a pair of indices $i_0^*, i_1^* \in [0, n_{\text{SSB-BLK}} - 1]$, and submits them to $\mathcal{C}$.
- $\mathcal{C}$ selects a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$ and computes $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}}, i_{\hat{b}}^*)$, and returns HK to $\mathcal{B}$.
- $\mathcal{B}$ eventually outputs a guess bit $\hat{b}' \in \{0,1\}$.

The SSB hash is said to be index hiding if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{SSB,IH}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► **Somewhere Statistically Binding**: An SSB hash key HK is said to be statistically binding for an index $i^* \in [0, n_{\text{SSB-BLK}} - 1]$ if there do not exist any $h \in \{0,1\}^{\ell_{\text{SSB-HASH}}}, u \neq u' \in \Sigma_{\text{SSB-BLK}}$, and $\pi_{\text{SSB}}, \pi'_{\text{SSB}} \in \Pi_{\text{SSB}}$ such that $\mathsf{SSB.Verify}(\text{HK}, h, i^*, u, \pi_{\text{SSB}}) = 1 = \mathsf{SSB.Verify}(\text{HK}, h, i^*, u', \pi'_{\text{SSB}})$.

The SSB hash is defined to be somewhere statistically binding if for any security parameter λ, integer $n_{\text{SSB-BLK}} \leq 2^\lambda$, index $i^* \in [0, n_{\text{SSB-BLK}} - 1]$, the hash key $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}}, i^*)$ is statistically binding for i^*. Note that this is an information theoretic property.

The first construction of an SSB hash is presented by Hubacek et al. [HW15]. Their construction is based on fully homomorphic encryption (FHE) [Gen09]. Recently, Okamoto et al. [OPWW15] provides alternative constructions of SSB hash based on various standard number theoretic assumptions.

In this paper, we will consider $\ell_{\text{SSB-BLK}} = 1$ and $n_{\text{SSB-BLK}} = 2^\lambda$.

2.4.2 Positional Accumulator

We will now present the notion of a positional accumulator as defined by Koppula et al. [KLW15].

Definition 2.4 (Positional Accumulator). A positional accumulator is comprised of the PPT algorithms (ACC.Setup, ACC.Setup-Enforce-Read, ACC.Setup-Enforce-Write, ACC.Prep-Read, ACC.Prep-Write, ACC.Verify-Read, ACC.Write-Store, ACC.Update) along with a block alphabet

$\Sigma_{\text{ACC-BLK}} = \{0,1\}^{\ell_{\text{ACC-BLK}}}$, accumulator size $\ell_{\text{ACC-ACCUMULATE}}$, proof space $\Pi_{\text{ACC}} = \{0,1\}^{\ell_{\text{ACC-PROOF}}}$ where $\ell_{\text{ACC-BLK}}, \ell_{\text{ACC-ACCUMULATE}}, \ell_{\text{ACC-PROOF}}$ are some polynomials in the security parameter λ. The algorithms have the following syntax:

$\mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}}) \rightarrow (\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0)$: The setup authority takes as input the security parameter 1^λ and an integer $n_{\text{ACC-BLK}} \leq 2^\lambda$ representing the maximum number of blocks that can be accumulated. It outputs the public parameters PP_{ACC}, an initial accumulator value w_0, and an initial storage value STORE_0.

$\mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}}, ((x_1, i_1), \ldots, (x_\kappa, i_\kappa)), i^*) \rightarrow (\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0)$: On input the security parameter 1^λ, an integer $n_{\text{ACC-BLK}} \leq 2^\lambda$ representing the maximum number of blocks that can be accumulated, a sequence of symbol-index pairs $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$, and an additional index $i^* \in [0, n_{\text{ACC-BLK}} - 1]$, the setup authority publishes the public parameters PP_{ACC}, an initial accumulator value w_0, together with an initial storage value STORE_0.

$\mathsf{ACC.Setup\text{-}Enforce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}}, ((x_1, i_1), \ldots, x_\kappa, i_\kappa))) \rightarrow (\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0)$: On input the security parameter 1^λ, an integer $n_{\text{ACC-BLK}} \leq 2^\lambda$ denoting the maximum number of blocks that can be accumulated, and a sequence of symbol-index pairs $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$, the setup authority publishes the public parameters PP_{ACC}, an initial accumulator value w_0, as well as, an initial storage value STORE_0.

$\mathsf{ACC.Prep\text{-}Read}(\text{PP}_{\text{ACC}}, \text{STORE}_{\text{IN}}, i_{\text{IN}}) \rightarrow (x_{\text{OUT}}, \pi_{\text{ACC}})$: A storage-maintaining party takes as input the public parameter PP_{ACC}, a storage value STORE_{IN}, and an index $i_{\text{IN}} \in [0, n_{\text{ACC-BLK}} - 1]$. It outputs a symbol $x_{\text{OUT}} \in \Sigma_{\text{ACC-BLK}} \cup \{\epsilon\}$ (ϵ being the empty string) and a proof $\pi_{\text{ACC}} \in \Pi_{\text{ACC}}$.

$\mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{\text{IN}}, i_{\text{IN}}) \rightarrow \text{AUX}$: Taking as input the public parameter PP_{ACC}, a storage value STORE_{IN}, together with an index $i_{\text{IN}} \in [0, n_{\text{ACC-BLK}} - 1]$, a storage-maintaining party outputs an auxiliary value AUX.

$\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, x_{\text{IN}}, i_{\text{IN}}, \pi_{\text{ACC}}) \rightarrow \hat{\beta} \in \{0,1\}$: A verifier takes as input the public parameter PP_{ACC}, an accumulator value $w_{\text{IN}} \in \{0,1\}^{\ell_{\text{ACC-ACCUMULATE}}}$, a symbol $x_{\text{IN}} \in \Sigma_{\text{ACC-BLK}} \cup \{\epsilon\}$, an index $i_{\text{IN}} \in [0, n_{\text{ACC-BLK}} - 1]$, and a proof $\pi_{\text{ACC}} \in \Pi_{\text{ACC}}$. It outputs outputs a bit $\hat{\beta} \in \{0,1\}$.

$\mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{\text{IN}}, i_{\text{IN}}, x_{\text{IN}}) \rightarrow \text{STORE}_{\text{OUT}}$: On input the public parameters PP_{ACC}, a storage value STORE_{IN}, an index $i_{\text{IN}} \in [0, n_{\text{ACC-BLK}} - 1]$, and a symbol $x_{\text{IN}} \in \Sigma_{\text{ACC-BLK}}$, a storage-maintaining party computes a new storage value $\text{STORE}_{\text{OUT}}$.

$\mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, x_{\text{IN}}, i_{\text{IN}}, \text{AUX}) \rightarrow w_{\text{OUT}}$ or $\perp$: An accumulator-updating party takes as input the public parameters PP_{ACC}, an accumulator value $w_{\text{IN}} \in \{0,1\}^{\ell_{\text{ACC-ACCUMULATE}}}$, a symbol $x_{\text{IN}} \in \Sigma_{\text{ACC-BLK}}$, an index $i_{\text{IN}} \in [0, n_{\text{ACC-BLK}} - 1]$, and an auxiliary value AUX. It outputs the updated accumulator value $w_{\text{OUT}} \in \{0,1\}^{\ell_{\text{ACC-ACCUMULATE}}}$ or the designated reject string $\perp$.

Following [KLW15,DKW16], we will consider the algorithms ACC.Setup, ACC.Setup-Enforce-Read, and ACC.Setup-Enforce-Write as randomized while all other algorithms as deterministic in this paper. The algorithms satisfy the following properties:

► **Correctness**: Consider any symbol-index pair sequence $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$. Fix any $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}})$. For $j = 1, \ldots, \kappa$, iteratively define the following:

- $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, i_j, x_j)$
- $\text{AUX}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, i_j)$

– $w_j = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}, w_{j-1}, x_j, i_j, \mathrm{AUX}_j)$

The following correctness properties are required to be satisfied:

i) For any security parameter λ, $n_{\text{ACC-BLK}} \leq 2^\lambda$, index $i^* \in [0, n_{\text{ACC-BLK}} - 1]$, sequence of symbol-index pairs $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$, and $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}})$, if STORE_κ is computed as above, then $\mathsf{ACC.Prep\text{-}Read}(\mathrm{PP}_{\mathrm{ACC}}, \mathrm{STORE}_\kappa, i^*)$ returns $(x_j, \pi_{\mathrm{ACC}})$ where j is the largest value in $[\kappa]$ such that $i_j = i^*$.
ii) For any security parameter λ, $n_{\text{ACC-BLK}} \leq 2^\lambda$, sequence of symbol-index pairs $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$, $i^* \in [0, n_{\text{ACC-BLK}} - 1]$, and $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}})$, if STORE_κ and w_κ are computed as described above and $(x_{\mathrm{OUT}}, \pi_{\mathrm{ACC}}) = \mathsf{ACC.Prep\text{-}Read}(\mathrm{PP}_{\mathrm{ACC}}, \mathrm{STORE}_\kappa, i^*)$, then $\mathsf{ACC.Verify\text{-}Read}(\mathrm{PP}_{\mathrm{ACC}}, w_\kappa, x_{\mathrm{OUT}}, i^*, \pi_{\mathrm{ACC}}) = 1$.

► **Indistinguishability of Read Setup**: This property of a positional accumulator is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ chooses a bound $n_{\text{ACC-BLK}} \leq 2^\lambda$ of the number of blocks, κ symbol-index pairs $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$, and an index $i^* \in [0, n_{\text{ACC-BLK}} - 1]$. It submits all of those to $\mathcal{C}$.
- $\mathcal{C}$ selects a random bit $\hat{b} \xleftarrow{\$} \{0, 1\}$. If $\hat{b} = 0$, $\mathcal{C}$ generates $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}})$. Otherwise, $\mathcal{C}$ generates $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}}, ((x_1, i_1), \ldots, (x_\kappa, i_\kappa)), i^*)$. It returns $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0)$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0, 1\}$.

The positional accumulator is said to satisfy indistinguishability of read setup if for any PPT adversary $\mathcal{B}$, for any security parameter λ, we have

$$\mathsf{Adv}_{\mathcal{B}}^{\text{ACC,IND-READ}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► **Indistinguishability of Write Setup**: This property of a positional accumulator is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ chooses a bound $n_{\text{ACC-BLK}} \leq 2^\lambda$ of the number of blocks and κ symbol-index pairs $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$. It submits all of those to $\mathcal{C}$.
- $\mathcal{C}$ selects a random bit $\hat{b} \xleftarrow{\$} \{0, 1\}$. If $\hat{b} = 0$, $\mathcal{C}$ generates $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}})$. Otherwise, $\mathcal{C}$ generates $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}}, ((x_1, i_1), \ldots, (x_\kappa, i_\kappa)))$. It returns $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0)$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0, 1\}$.

A positional accumulator is said to satisfy indistinguishability of write setup if for any PPT adversary $\mathcal{B}$, for any security parameter λ, we have

$$\mathsf{Adv}_{\mathcal{B}}^{\text{ACC,IND-WRITE}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► **Read Enforcing**: Consider any security parameter λ, $n_{\text{ACC-BLK}} \leq 2^\lambda$, $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$, and $i^* \in [0, n_{\text{ACC-BLK}} - 1]$.

Let $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}}, ((x_1, i_1), \ldots, (x_\kappa, i_\kappa)), i^*)$. For $j = 1, \ldots, \kappa$, iteratively define the following:

– $\mathrm{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\mathrm{PP}_{\mathrm{ACC}}, \mathrm{STORE}_{j-1}, i_j, x_j)$

– $\text{AUX}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, i_j)$
– $w_j = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{j-1}, x_j, i_j, \text{AUX}_j)$

The positional accumulator is said to be read enforcing if $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_\kappa, x_{\text{IN}}, i^*, \pi_{\text{ACC}}) = 1$ implies either $[i^* \notin \{i_1, \ldots, i_\kappa\}] \wedge [x_{\text{IN}} = \epsilon]$ or $x_{\text{IN}} = x_j$ for the largest $j \in [\kappa]$ such that $i_j = i^*$. Note that this is an information theoretic property.

► **Write Enforcing**: Consider any security parameter λ, $n_{\text{ACC-BLK}} \leq 2^\lambda$, and $((x_1, i_1), \ldots, (x_\kappa, i_\kappa)) \in (\Sigma_{\text{ACC-BLK}} \times [0, n_{\text{ACC-BLK}} - 1])^\kappa$. Let $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}}, ((x_1, i_1), \ldots, (x_\kappa, i_\kappa)))$. For $j = 1, \ldots, \kappa$, iteratively define the following:

– $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, i_j, x_j)$
– $\text{AUX}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, i_j)$
– $w_j = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{j-1}, x_j, i_j, \text{AUX}_j)$

The positional accumulator is defined to be write enforcing if $\mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\kappa-1}, x_\kappa, i_\kappa, \text{AUX}) = w_{\text{OUT}} \neq \perp$, for any AUX, implies $w_{\text{OUT}} = w_\kappa$. Observe that this is an information theoretic property as well.

The first construction of a positional accumulator is presented by Koppula et al. [KLW15] based on IO and one-way function. Recently, Okamoto et al. [OPWW15] provided an alternative construction of positional accumulator from standard number theoretic assumptions.

2.4.3 Iterator

Here we describe the concept of a cryptographic iterator again following Koppula et al. [KLW15]. In the same paper, a construction of the primitive from IO and one-way function is demonstrated.

Definition 2.5 (Iterator). A cryptographic iterator consists of the PPT algorithms (ITR.Setup, ITR.Set-Enforce, ITR.Iterate) along with a message space $\mathcal{M}_{\text{ITR}} = \{0,1\}^{\ell_{\text{ITR-MSG}}}$ and iterator state size $\ell_{\text{ITR-ST}}$, where $\ell_{\text{ITR-MSG}}, \ell_{\text{ITR-ST}}$ are some polynomials in the security parameter λ. The algorithms have the following syntax:

$\mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}}) \rightarrow (\text{PP}_{\text{ITR}}, v_0)$: The setup authority takes as input the security parameter 1^λ along with an integer bound $n_{\text{ITR}} \leq 2^\lambda$ on the number of iterations. It outputs the public parameters PP_{ITR} and an initial state $v_0 \in \{0,1\}^{\ell_{\text{ITR-ST}}}$.

$\mathsf{ITR.Setup\text{-}Enforce}(1^\lambda, n_{\text{ITR}}, (\mu_1, \ldots, \mu_\kappa)) \rightarrow (\text{PP}_{\text{ITR}}, v_0)$: Taking as input the security parameter 1^λ, an integer bound $n_{\text{ITR}} \leq 2^\lambda$, together with a sequence of κ messages $(\mu_1, \ldots, \mu_\kappa) \in \mathcal{M}_{\text{ITR}}^\kappa$, where $\kappa \leq n_{\text{ITR}}$, the setup authority publishes the public parameters PP_{ITR} and an initial state $v_0 \in \{0,1\}^{\ell_{\text{ITR-ST}}}$.

$\mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}} \in \{0,1\}^{\ell_{\text{ITR-ST}}}, \mu) \rightarrow v_{\text{OUT}}$: On input the public parameters PP_{ITR}, a state v_{IN}, and a message $\mu \in \mathcal{M}_{\text{ITR}}$, an iterator outputs an updated state $v_{\text{OUT}} \in \{0,1\}^{\ell_{\text{ITR-ST}}}$. For any integer $\kappa \leq n_{\text{ITR}}$, we will write $\mathsf{ITR.Iterate}^\kappa(\text{PP}_{\text{ITR}}, v_0, (\mu_1, \ldots, \mu_\kappa))$ to denote $\mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\kappa-1}, \mu_\kappa)$, where v_j is defined iteratively as $v_j = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{j-1}, \mu_j)$ for all $j = 1, \ldots, \kappa - 1$.

The algorithm ITR.Iterate is deterministic, while the other two algorithms are randomized. The algorithms satisfy the following properties:

► **Indistinguishability of Enforcing Setup**: This property of a cryptographic iterator is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ chooses an integer bound $n_{\text{ITR}} \leq 2^\lambda$, along with a sequence of κ messages $(\mu_1, \ldots, \mu_\kappa) \in \mathcal{M}_{\text{ITR}}^\kappa$, and submits them to $\mathcal{C}$.
- $\mathcal{C}$ selects a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. If $\hat{b} = 0$, $\mathcal{C}$ generates $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}})$. Else, $\mathcal{C}$ generates $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup\text{-}Enforce}(1^\lambda, n_{\text{ITR}}, (\mu_1, \ldots, \mu_\kappa))$. It sends back $(\text{PP}_{\text{ITR}}, v_0)$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The cryptographic iterator is said to satisfy indistinguishability of enforcing setup if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{ITR,IND-ENF}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► **Enforcing**: Consider any security parameter λ, $n_{\text{ITR}} \leq 2^\lambda$, $\kappa \leq n_{\text{ITR}}$, and $(\mu_1, \ldots, \mu_\kappa) \in \mathcal{M}_{\text{ITR}}^\kappa$. Let $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Set\text{-}Enforce}(1^\lambda, n_{\text{ITR}}, (\mu_1, \ldots, \mu_\kappa))$ and $v_j = \mathsf{ITR.Iterate}^j(\text{PP}_{\text{ITR}}, v_0, (\mu_1, \ldots, \mu_j))$ for all $j \in [\kappa]$. The cryptographic iterator is said to be enforcing if $v_k = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v', \mu')$ implies $(v', \mu') = (v_{\kappa-1}, \mu_\kappa)$. Note that this is an information theoretic property.

2.4.4 Splittable Signature

The following background on splittable signatures is taken verbatim from Koppula et al. [KLW15] as well.

Definition 2.6 (Splittable Signature: SPS). A splittable signature scheme (SPS) for message space $\mathcal{M}_{\text{SPS}} = \{0,1\}^{\ell_{\text{SPS-MSG}}}$ and signature space $\mathcal{S}_{\text{SPS}} = \{0,1\}^{\ell_{\text{SPS-SIG}}}$, where $\ell_{\text{SPS-MSG}}, \ell_{\text{SPS-SIG}}$ are some polynomials in the security parameter λ, consists of PPT algorithms (SPS.Setup, SPS.Sign, SPS.Verify, SPS.Split, SPS.Sign-ABO) which are described below:

$\mathsf{SPS.Setup}(1^\lambda) \to (\text{SK}_{\text{SPS}}, \text{VK}_{\text{SPS}}, \text{VK}_{\text{SPS-REJ}})$: The setup authority takes as input the security parameter 1^λ and generates a signing key SK_{SPS}, a verification key VK_{SPS}, together with a reject verification key $\text{VK}_{\text{SPS-REJ}}$.

$\mathsf{SPS.Sign}(\text{SK}_{\text{SPS}}, m) \to \sigma_{\text{SPS}}$: A signer given a signing key SK_{SPS} along with a message $m \in \mathcal{M}_{\text{SPS}}$, produces a signature $\sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}}$.

$\mathsf{SPS.Verify}(\text{VK}_{\text{SPS}}, m, \sigma_{\text{SPS}}) \to \hat{\beta} \in \{0,1\}$: A verifier takes as input a verification key VK_{SPS}, a message $m \in \mathcal{M}_{\text{SPS}}$, and a signature $\sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}}$. It outputs a bit $\hat{\beta} \in \{0,1\}$.

$\mathsf{SPS.Split}(\text{SK}_{\text{SPS}}, m^*) \to (\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}})$: On input a signing key SK_{SPS} along with a message $m^* \in \mathcal{M}_{\text{SPS}}$, the setup authority generates a signature $\sigma_{\text{SPS-ONE},m^*} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS}}, m^*)$, a one-message verification key $\text{VK}_{\text{SPS-ONE}}$, and all-but-one signing-verification key pair $(\text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}})$.

$\mathsf{SPS.Sign\text{-}ABO}(\text{SK}_{\text{SPS-ABO}}, m) \to \sigma_{\text{SPS}}$ or $\perp$: An all-but-one signer given an all-but-one signing key $\text{SK}_{\text{SPS-ABO}}$ and a message $m \in \mathcal{M}_{\text{SPS}}$, outputs a signature $\sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}}$ or a distinguished string $\perp$ to indicate failure. For simplicity of notation, we will often use $\mathsf{SPS.Sign}(\text{SK}_{\text{SPS-ABO}}, m)$ to represent the output of this algorithm.

We note that among the algorithms described above, SPS.Setup and SPS.Split are randomized while all the others are deterministic. The algorithms satisfy the following properties:

► **Correctness**: For any security parameter λ, message $m^* \in \mathcal{M}_{\text{SPS}}$, $(\text{SK}_{\text{SPS}}, \text{VK}_{\text{SPS}}, \text{VK}_{\text{SPS-REJ}}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$, and $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}_{\text{SPS}}, m^*)$ the following correctness conditions hold:

i) $\forall\, m \in \mathcal{M}_{\text{SPS}}, \mathsf{SPS.Verify}(\text{VK}_{\text{SPS}}, m, \mathsf{SPS.Sign}(\text{SK}_{\text{SPS}}, m)) = 1.$
ii) $\forall\, m \neq m^* \in \mathcal{M}_{\text{SPS}}, \mathsf{SPS.Sign}(\text{SK}_{\text{SPS}}, m) = \mathsf{SPS.Sign\text{-}ABO}(\text{SK}_{\text{SPS-ABO}}, m).$
iii) $\forall\, \sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}}, \mathsf{SPS.Verify}(\text{VK}_{\text{SPS-ONE}}, m^*, \sigma_{\text{SPS}}) = \mathsf{SPS.Verify}(\text{VK}_{\text{SPS}}, m^*, \sigma_{\text{SPS}}).$
iv) $\forall\, m \neq m^* \in \mathcal{M}_{\text{SPS}}, \sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}}, \mathsf{SPS.Verify}(\text{VK}_{\text{SPS-ABO}}, m, \sigma_{\text{SPS}}) = \mathsf{SPS.Verify}(\text{VK}_{\text{SPS}}, m, \sigma_{\text{SPS}}).$
v) $\forall\, m \neq m^* \in \mathcal{M}_{\text{SPS}}, \sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}}, \mathsf{SPS.Verify}(\text{VK}_{\text{SPS-ONE}}, m, \sigma_{\text{SPS}}) = 0.$
vi) $\forall\, \sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}}, \mathsf{SPS.Verify}(\text{VK}_{\text{SPS-ABO}}, m^*, \sigma_{\text{SPS}}) = 0.$
vii) $\forall\, m \in \mathcal{M}_{\text{SPS}}, \sigma_{\text{SPS}} \in \mathcal{S}_{\text{SPS}}, \mathsf{SPS.Verify}(\text{VK}_{\text{SPS-REJ}}, m, \sigma_{\text{SPS}}) = 0.$

► $\text{VK}_{\text{SPS-REJ}}$ **Indistinguishability**: This property of a splittable signature scheme is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{C}$ generates $(\text{SK}_{\text{SPS}}, \text{VK}_{\text{SPS}}, \text{VK}_{\text{SPS-REJ}}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$. Next it chooses a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. If $\hat{b} = 0$, it sends VK_{SPS} to $\mathcal{B}$. Otherwise, it sends $\text{VK}_{\text{SPS-REJ}}$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The splittable signature scheme is said to be $\text{VK}_{\text{SPS-REJ}}$ indistinguishable if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{SPS,IND-REJ}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► $\text{VK}_{\text{SPS-ONE}}$ **Indistinguishability**: This feature of a splittable signature scheme is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ submits a message $m^* \in \mathcal{M}_{\text{SPS}}$ to $\mathcal{C}$.
- $\mathcal{C}$ generates $(\text{SK}_{\text{SPS}}, \text{VK}_{\text{SPS}}, \text{VK}_{\text{SPS-REJ}}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$. Next it computes $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}_{\text{SPS}}, m^*)$. Then it chooses a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. If $\hat{b} = 0$, it returns $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}})$ to $\mathcal{B}$. Else, it returns $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS}})$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The splittable signature scheme is said to be $\text{VK}_{\text{SPS-ONE}}$ indistinguishable if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{SPS,IND-ONE}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► $\text{VK}_{\text{SPS-ABO}}$ **Indistinguishability**: This feature of a splittable signature scheme is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ submits a message $m^* \in \mathcal{M}_{\text{SPS}}$ to $\mathcal{C}$.
- $\mathcal{C}$ generates $(\text{SK}_{\text{SPS}}, \text{VK}_{\text{SPS}}, \text{VK}_{\text{SPS-REJ}}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$. Next it computes $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}_{\text{SPS}}, m^*)$. Then it chooses a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. If $\hat{b} = 0$, it returns $(\text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}})$ to $\mathcal{B}$. Else, it returns $(\text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS}})$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The splittable signature scheme is said to be $\text{VK}_{\text{SPS-ABO}}$ indistinguishable if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{SPS,IND-ABO}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

► **Splitting Indistinguishability**: This feature of a splittable signature scheme is defined through the following experiment between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$:

- $\mathcal{B}$ submits a message $m^* \in \mathcal{M}_{\text{SPS}}$ to $\mathcal{C}$.
- $\mathcal{C}$ forms $(\text{SK}_{\text{SPS}}, \text{VK}_{\text{SPS}}, \text{VK}_{\text{SPS-REJ}}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda), (\text{SK}'_{\text{SPS}}, \text{VK}'_{\text{SPS}}, \text{VK}'_{\text{SPS-REJ}}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$. Next it computes $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}_{\text{SPS}}, m^*)$ as well as $(\sigma'_{\text{SPS-ONE},m^*}, \text{VK}'_{\text{SPS-ONE}}, \text{SK}'_{\text{SPS-ABO}}, \text{VK}'_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}'_{\text{SPS}}, m^*)$. Then it chooses a random bit $\hat{b} \xleftarrow{\$} \{0,1\}$. If $\hat{b} = 0$, it returns $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}})$ to $\mathcal{B}$. Else, it returns $(\sigma_{\text{SPS-ONE},m^*}, \text{VK}_{\text{SPS-ONE}}, \text{SK}'_{\text{SPS-ABO}}, \text{VK}'_{\text{SPS-ABO}})$ to $\mathcal{B}$.
- $\mathcal{B}$ outputs a guess bit $\hat{b}' \in \{0,1\}$.

The splittable signature scheme is said to be splitting indistinguishable if for any PPT adversary $\mathcal{B}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{SPS,IND-SPL}}(\lambda) = |\mathsf{Pr}[\hat{b} = \hat{b}'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

Koppula et al. [KLW15] have constructed a splittable signature scheme using IO and one-way function.

3 Our Constrained Pseudorandom Function for Turing Machines

3.1 Notion

We start by formally defining the notion of constrained pseudorandom functions (CPRF's) following Deshpande et al. [DKW16]. Informally, a CPRF extends the idea of standard pseudorandom functions (PRF), enabling the master PRF key holder to generate 'constrained keys' that allow PRF evaluation on certain inputs, while the PRF evaluation on remaining inputs 'looks' random to any computationally bounded party holding only a constrained key. As in [DKW16], in order to allow unbounded polynomial length inputs, we associate the constrained keys to polynomial-time TM's.

Definition 3.1 (Constrained Pseudorandom Function for Turing Machines: CPRF). Let $\mathbb{M}_\lambda$ be a family of TM's with (worst case) running time bounded by $T = 2^\lambda$. A constrained pseudorandom function (CPRF) with key space $\mathcal{K}_{\text{CPRF}}$, input domain $\mathcal{X}_{\text{CPRF}} \subset \{0,1\}^*$, and output space $\mathcal{Y}_{\text{CPRF}} \subset \{0,1\}^*$ for the TM family $\mathbb{M}_\lambda$ consists of an additional key space $\mathcal{K}_{\text{CPRF-CONST}}$ and PPT algorithms (CPRF.Setup, CPRF.Eval, CPRF.Constrain, CPRF.Eval-Constrained) described as follows:

$\mathsf{CPRF.Setup}(1^\lambda) \to \text{SK}_{\text{CPRF}}$: The setup authority takes as input the security parameter 1^λ and generates the master CPRF key $\text{SK}_{\text{CPRF}} \in \mathcal{K}_{\text{CPRF}}$.

$\mathsf{CPRF.Eval}(\text{SK}_{\text{CPRF}}, x) \to y$: On input the master CPRF key SK_{CPRF} along with an input $x \in \mathcal{X}_{\text{CPRF}}$, the setup authority computes the value of the CPRF $y \in \mathcal{Y}_{\text{CPRF}}$. For simplicity of notation, we will use $\mathsf{CPRF}(\text{SK}_{\text{CPRF}}, x)$ to indicate the output of this algorithm.

$\mathsf{CPRF.Constrain}(\text{SK}_{\text{CPRF}}, M) \to \text{SK}_{\text{CPRF}}\{M\}$: Taking as input the master CPRF key SK_{CPRF} and a TM $M \in \mathbb{M}_\lambda$, the setup authority provides a constrained key $\text{SK}_{\text{CPRF}}\{M\} \in \mathcal{K}_{\text{CPRF-CONST}}$ to a legitimate user.

$\mathsf{CPRF.Eval\text{-}Constrained}(\text{SK}_{\text{CPRF}}\{M\}, x) \to y$ or $\perp$: A user takes as input a constrained key $\text{SK}_{\text{CPRF}}\{M\} \in \mathcal{K}_{\text{CPRF-CONST}}$, corresponding to a legitimate TM $M \in \mathbb{M}_\lambda$, along with an input $x \in \mathcal{X}_{\text{CPRF}}$. It outputs either a value $y \in \mathcal{Y}_{\text{CPRF}}$ or $\perp$ indicating failure.

The algorithms $\mathsf{CPRF.Setup}$ and $\mathsf{CPRF.Constrain}$ are randomized, whereas, the other two are deterministic. The algorithms satisfy the following properties:

► **Correctness under Constraining**: Consider any security parameter λ, $\mathrm{SK}_{\mathrm{CPRF}} \in \mathcal{K}_{\mathrm{CPRF}}$, $M \in \mathbb{M}_\lambda$, and $\mathrm{SK}_{\mathrm{CPRF}}\{M\} \xleftarrow{\$} \mathsf{CPRF.Constrain}(\mathrm{SK}_{\mathrm{CPRF}}, M)$. The following must hold:

$$\mathsf{CPRF.Eval\text{-}Constrained}(\mathrm{SK}_{\mathrm{CPRF}}\{M\}, x) = \begin{cases} \mathsf{CPRF}(\mathrm{SK}_{\mathrm{CPRF}}, x), \text{ if } M(x) = 1 \\ \perp, \qquad\qquad\qquad\quad \text{otherwise} \end{cases}$$

► **Selective Pseudorandomness**: This property of a CPRF is defined through the following experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$:

- $\mathcal{A}$ submits a challenge input $x^* \in \mathcal{X}_{\mathrm{CPRF}}$ to $\mathcal{B}$.
- $\mathcal{B}$ generates a master CPRF key $\mathrm{SK}_{\mathrm{CPRF}} \xleftarrow{\$} \mathsf{CPRF.Setup}(1^\lambda)$. Next it selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, it computes $y^* = \mathsf{CPRF}(\mathrm{SK}_{\mathrm{CPRF}}, x^*)$. Otherwise, it chooses a random $y^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{CPRF}}$. It returns y^* to $\mathcal{A}$.
- $\mathcal{A}$ may adaptively make a polynomial number of queries of the following kinds to $\mathcal{B}$:
  - **Evaluation query**: $\mathcal{A}$ queries the CPRF value at some input $x \in \mathcal{X}_{\mathrm{CPRF}}$ such that $x \neq x^*$. $\mathcal{B}$ provides the CPRF value $\mathsf{CPRF}(\mathrm{SK}_{\mathrm{CPRF}}, x)$ to $\mathcal{A}$.
  - **Key query**: $\mathcal{A}$ queries a constrained key corresponding to TM $M \in \mathbb{M}_\lambda$ subject to the constraint that $M(x^*) = 0$. $\mathcal{B}$ gives the constrained key $\mathrm{SK}_{\mathrm{CPRF}}\{M\} \xleftarrow{\$} \mathsf{CPRF.Constrain}(\mathrm{SK}_{\mathrm{CPRF}}, M)$ to $\mathcal{A}$.
- $\mathcal{A}$ eventually outputs a guess bit $b' \in \{0,1\}$.

The CPRF is said to be selectively pseudorandom if for any PPT adversary $\mathcal{A}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{A}}^{\mathrm{CPRF,SEL\text{-}PR}}(\lambda) = |\mathsf{Pr}[b = b'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

Remark 3.1. As pointed out in [HKKW14, CRV14], note that in the above selective pseudorandomness experiment, without loss of generality we may assume that the adversary $\mathcal{A}$ only makes constrained key queries and no evaluation query. This is because any evaluation query at input $x \in \mathcal{X}_{\mathrm{CPRF}}$ can be replaced by constrained key query for a TM $M_x \in \mathbb{M}_\lambda$ that accepts only x. Since, the restriction on the evaluation queries is that $x \neq x^*$, $M_x(x^*) = 0$, and thus M_x is a valid constrained key query. We will use this simplification in our proof.

3.2 The CPRF Construction of Deshpande et al.

In EUROCRYPT 2016, Deshpande et al. [DKW16] have presented a CPRF construction supporting inputs of unconstrained polynomial length based on indistinguishability obfuscation and injective pseudorandom generators, which they have claimed to be selectively secure.Unfortunately, their security argument has a flaw. In this section, we give an informal description of their CPRF construction and point out the flaw in their security argument.

Overview of the CPRF Construction of [DKW16]: The principle ideas behind the CPRF construction of [DKW16] are as follows: To produce the CPRF output their construction uses a PPRF $\mathcal{F}$ and a positional accumulator. A master CPRF key consists of a key K for the PPRF $\mathcal{F}$ and a set of public parameters $\mathrm{PP}_{\mathrm{ACC}}$ of the positional accumulator. The CPRF evaluation on some input $x = x_0 \ldots x_{\ell_x - 1} \in \mathcal{X}_{\mathrm{CPRF}} \subset \{0,1\}^*$ is simply $\mathcal{F}(K, w_{\mathrm{INP}})$, where w_{INP} is the accumulation of the bits of x using $\mathrm{PP}_{\mathrm{ACC}}$.

A constrained key of the CPRF, corresponding to some TM M, comprises of PP_{ACC} along with two programs $\mathcal{P}_1$ and $\mathcal{P}_{\text{CPRF}}$, which are obfuscated using IO. The first program $\mathcal{P}_1$, also known as the *initial signing program*, takes as input an accumulator value and outputs a signature on it together with the initial state and header position of the TM M. The second program $\mathcal{P}_{\text{CPRF}}$, also called the *next step program*, takes as input a state and header position of M along with an input symbol and an accumulator value. It essentially computes the next step function of M on the input state-symbol pair, and eventually outputs the proper PRF value, if M reaches the accepting state. The program $\mathcal{P}_{\text{CPRF}}$ also performs certain authenticity checks before computing the next step function of M in order to prevent illegal inputs. For this purpose, $\mathcal{P}_{\text{CPRF}}$ additionally takes as input a signature on the input state, header position, and accumulator value, together with a proof for the positional accumulator. The program $\mathcal{P}_{\text{CPRF}}$ verifies the signature as well as checks the accumulator proof to get convinced that the input symbol is indeed the one placed at the input header position of the underlying storage of the input accumulator value. If all these verifications pass, then $\mathcal{P}_{\text{CPRF}}$ determines the next state and header position of M, as well as, the new symbol that needs to be written to the input header position. The program $\mathcal{P}_{\text{CPRF}}$ then updates the accumulator value by placing the new symbol at the input header position as well as signs the updated accumulator value along with the computed next state and header position of M. The signature scheme used by the two programs is a splittable signature. In order to deal with the positional accumulator related verifications and updations, the program $\mathcal{P}_{\text{CPRF}}$ has PP_{ACC} hardwired.

Evaluating the CPRF on some input x using a constrained key, corresponding to some TM M, consists of two steps. In the first step, the evaluator computes the accumulation w_{INP} of the bits of x using PP_{ACC}, which are also included in the constrained key, and then obtains a signature on w_{INP} together with the initial state and header position of M by running the program $\mathcal{P}_1$. The second step is to repeatedly run the program $\mathcal{P}_{\text{CPRF}}$, each time on input the current accumulator value, current state and header position of M, along with the signature on them. Additionally, in each iteration the evaluator also feeds w_{INP} to $\mathcal{P}_{\text{CPRF}}$. The iteration is continued until the program $\mathcal{P}_{\text{CPRF}}$ either outputs the PRF evaluation or the designated null string $\perp$ indicating failure.

The Flaw: In order to prove selective pseudorandomness of the above CPRF construction, the authors of [DKW16] extends the techniques introduced in [KLW15] in the context of proving security of message-hiding encoding scheme for TM's. More precisely, the authors of [DKW16] proceed as follows: During the course of the proof, the authors aim to modify the constrained keys given to the adversary $\mathcal{A}$ in the selective pseudorandomness experiment, discussed in Section 3.1, to embed the punctured PPRF key $K\{w^*_{\text{INP}}\}$ punctured at w^*_{INP} instead of the full PPRF key K, which is part of the master CPRF key sampled by the challenger $\mathcal{B}$. Here, w^*_{INP} is the accumulation of the bits of the challenge input x^*, submitted by the adversary $\mathcal{A}$, using PP_{ACC}, included within the master CPRF key generated by the challenger $\mathcal{B}$. In order to make this substitution, it is to be ensured that the obfuscated next step programs included in the constrained keys never outputs the PRF evaluation for inputs corresponding to w^*_{INP} even if reaching the accepting state. The proof transforms the constrained keys one at a time through multiple hybrid steps. Suppose that the total number of constrained keys queried by $\mathcal{A}$ be $\hat{q}$. Consider the transformation of the ν^{th} constrained key ($1 \leq \nu \leq \hat{q}$) corresponding to the TM $M^{(\nu)}$ that runs on the challenge input x^* for $t^{*(\nu)}$ steps and reaches the rejecting state. In the course of transformation, the obfuscated next step program $\mathcal{P}^{(\nu)}_{\text{CPRF}}$ of the ν^{th} constrained key is first altered to one that never outputs the PRF evaluation for inputs corresponding to w^*_{INP} within the first $t^{*(\nu)}$ steps. Towards accomplishing this transition, the challenger $\mathcal{B}$ at various stages needs to generate PP_{ACC} in read/write enforcing mode where the enforcing property should be tailored to the steps of execution of the specific TM $M^{(\nu)}$ on x^*. For instance, at some point of transformation of the ν^{th} constrained key, PP_{ACC} needs to be set in the read enforcing mode by $\mathcal{B}$ on input (i) the entire sequence of symbol-position pairs arising from iteratively running $M^{(\nu)}$ on x^* upto the t^{th} step and (ii) the enforcing

index corresponding to the header position of $M^{(\nu)}$ at the t^{th} step while running on x^*, where $1 < t \leq t^{*(\nu)}$. Evidently, $\mathcal{B}$ can determine those symbol-position pairs *only after receiving* the TM $M^{(\nu)}$ from $\mathcal{A}$. However, $\mathcal{B}$ would also require PP_{ACC} while creating the constrained keys queried by $\mathcal{A}$ before making the ν^{th} constrained key query and even possibly for preparing the challenge value for $\mathcal{A}$. Thus, it is immediate that $\mathcal{B}$ must generate PP_{ACC} *prior to receiving* the ν^{th} query from $\mathcal{A}$. This is *impossible* as setting PP_{ACC} in read enforcing mode requires the knowledge of the TM $M^{(\nu)}$, which is *not available* before the ν^{th} constrained key query of $\mathcal{A}$. A similar conflict also arises when $\mathcal{B}$ attempts to setup PP_{ACC} in the write enforcing mode tailored to $M^{(\nu)}$. This serious flaw renders the security argument of [DKW16] invalid.

3.3 Overview of Our Techniques to Fix the Flaw of [DKW16]

Observe that a set of public parameters of the positional accumulator must be included within each constrained key. This is mandatory due to the required updatability feature of positional accumulator, which is indispensable to keep track of the current situation while running the obfuscated next step program $\mathcal{P}_{\text{CPRF}}$ iteratively in the course of evaluating the CPRF on some input. The root cause of the problem in the security argument of [DKW16] is the use of a single set of public parameters PP_{ACC} of the positional accumulator throughout the system. Therefore, as a first step, we attempt to assign a fresh set of public parameters of the positional accumulator to each constrained key. However, for compressing the PRF input to a fixed length, on which $\mathcal{F}$ can be applied producing the PRF output, we need a system-wide compressing tool. We employ SSB hash for this purpose. The idea is that while evaluating the CPRF on some input x using a constrained key, corresponding to some TM M, the evaluator first computes the hash value h by hashing x using the system wide SSB hash key, which is part of the master key. The evaluator also computes the accumulator value w_{INP} by accumulating the bits of x using the public parameters of positional accumulator included in the constrained key. Then, using the obfuscated initial signing program $\mathcal{P}_1$, included in the constrained key, the evaluator will obtain a signature on w_{INP} along with the initial state and header position of M. Finally, the evaluator will repeatedly run the obfuscated next step program $\mathcal{P}_{\text{CPRF}}$, included in the constrained key, each time giving as input all the quantities as in the evaluation algorithm of [DKW16], except that it now feeds the SSB hash value h in place of w_{INP} in each iteration. This is because, in case $\mathcal{P}_{\text{CPRF}}$ reaches the accepting state, it would require h to apply $\mathcal{F}$ for producing the PRF output.

However, this approach is not completely sound yet. Observe that, a possibly malicious evaluator can compute the SSB hash value h on the input x, on which it wishes to evaluate the CPRF although M does not accepts it, and initiates the evaluation by accumulating the bits of only a substring of x or some entirely different input, which is accepted by M. To prevent such malicious behavior, we include another IO-obfuscated program $\mathcal{P}_2$ within the constrained key, known as the *accumulating program*, whose purpose is to *restrict* the evaluator from accumulating the bits of a different input rather than the hashed one. The program $\mathcal{P}_2$ takes as input an SSB hash value h, an index i, a symbol, an accumulator value, a signature on the input accumulator value (along with the initial state and header position of M), and an opening value for SSB. The program $\mathcal{P}_2$ verifies the signature and also checks whether the input symbol is indeed present at the index i of the string that has been hashed to form h, using the input opening value. If all of these verifications pass, then $\mathcal{P}_2$ updates the input accumulator value by writing the input symbol at the i^{th} position of the accumulator storage. We also modify the obfuscated initial signing program $\mathcal{P}_1$, included in the constrained key, to take as input a hash value and output a signature on the accumulator value corresponding to the empty accumulator storage, along with the initial state and header position of M.

Moreover, for forbidding the evaluator from performing the evaluation by accumulating an M-accepted substring of the hashed input, we define our PRF output as the evaluation of $\mathcal{F}$ on the pair (hash value, length) of the input in stead of just the hash value of the input. Note that,

without loss of generality, we can set the upper bound of the length of PRF inputs to be 2^λ, where λ is the underlying security parameter in view of the fact that by suitably choosing λ we can accommodate inputs of any polynomial length. This setting of upper bound on the input length is implicitly considered in [DKW16] and also explicitly used in [BGJS15] while dealing with multi-input functional encryption for unbounded arity functions. Now, as the input length is bounded by 2^λ, the input length can be expressed as a bit strings of length λ. Thus, the PRF input length can be safely fed along with the SSB hash value of PRF input to $\mathcal{F}$, which can handle only inputs of apriori bounded length. Hence, the obfuscated next step programs $\mathcal{P}_{\text{CPRF}}$ included in our constrained keys must also take as input the length of the PRF input for producing the PRF value if reaching to the accepting state.

Therefore, to evaluate the CPRF on some input using a constrained key, corresponding to some TM M, an evaluator first hash the PRF input. The evaluator also obtains a signature on the empty accumulator value included in the constrained key, by running the obfuscated initial signing program $\mathcal{P}_1$ on input the computed hash value. Next, it repeatedly runs the obfuscated accumulating program $\mathcal{P}_2$ to accumulate the bits of the PRF input. Finally, it runs the obfuscated next step program $\mathcal{P}_{\text{CPRF}}$ iteratively on the current accumulator value along with other legitimate inputs until it obtains either the PRF output or $\perp$.

Regarding the proof of security, notice that the problem with enforcing the public parameters of the positional accumulator while transforming the queried constrained keys will not appear in our case as we have assigned a separate set of public parameters of positional accumulator to each constrained key. However, our actual security proof involves many subtleties that are difficult to describe with this high level description and is provided in full details in the sequel. We would only like to mention here that to cope up with certain issues in the proof we further include another IO-obfuscated program $\mathcal{P}_3$ in the constrained keys, known as the *signature changing program*, that changes the signature on the accumulation of the bits of the PRF input before starting the iterative computation with the obfuscated next step program $\mathcal{P}_{\text{CPRF}}$.

We follow the same novel technique introduced in [DKW16] for handling the tail hybrids in the final stage of transformation of the constrained keys. Note that as in [DKW16], we are also considering TM's which run for at most $T = 2^\lambda$ steps on any input. Unlike [KLW15], the authors of [DKW16] have devised a beautiful approach to obtain an end to end polynomial reduction to the security of IO for the tail hybrids by means of an injective pseudorandom generator (PRG). We directly adopt that technique to deal with the tail hybrids in our security proof. A high level overview of the approach is sketched below. Let us call the time step 2^τ as the τ^{th} landmark and the interval $[2^\tau, 2^{\tau+1} - 1]$ as the τ^{th} interval. Like [DKW16], our obfuscated next step programs $\mathcal{P}_{\text{CPRF}}$ included within the constrained keys take an additional PRG seed as input at each time step, and perform some additional checks on the input PRG seed. At time steps just before a landmark, the programs output a new pseudorandomly generated PRG seed, which is then used in the next interval. Using standard IO techniques, it can be shown that for inputs corresponding to (h^*, ℓ^*), if the program $\mathcal{P}_{\text{CPRF}}$ outputs $\perp$, for all time steps upto the one just before a landmark, then we can alter the program indistinguishably so that it outputs $\perp$ at all time steps in the next interval. Here h^* and ℓ^* are respectively the SSB hash value and length of the challenge input x^* submitted by the adversary $\mathcal{A}$ in the selective pseudorandomness experiment. Employing this technique, we can move across an exponential number of time steps at a single switch of the next step program $\mathcal{P}_{\text{CPRF}}$.

3.4 Formal Description of Our CPRF Construction

Now we will formally present our CPRF construction where the constrained keys are associated with TM's. Let λ be the underlying security parameter. Consider the family $\mathbb{M}_\lambda$ of TM's, the members of which have (worst-case) running time bounded by $T = 2^\lambda$, input alphabet

$\Sigma_{\text{INP}} = \{0,1\}$, and tape alphabet $\Sigma_{\text{TAPE}} = \{0,1,\_\}$. Our CPRF construction utilizes the following cryptographic building blocks:

i) $\mathcal{IO}$: An indistinguishability obfuscator for general polynomial-size circuits.
ii) $\mathsf{SSB} = (\mathsf{SSB.Gen}, \mathcal{H}, \mathsf{SSB.Open}, \mathsf{SSB.Verify})$: A somewhere statistically binding hash function with $\Sigma_{\text{SSB-BLK}} = \{0,1\}$.
iii) $\mathsf{ACC} = (\mathsf{ACC.Setup}, \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}, \mathsf{ACC.Setup\text{-}Enforce\text{-}Write}, \mathsf{ACC.Prep\text{-}Read}, \mathsf{ACC.Prep\text{-}Write}, \mathsf{ACC.Verify\text{-}Read}, \mathsf{ACC.Write\text{-}Store}, \mathsf{ACC.Update})$: A positional accumulator with $\Sigma_{\text{ACC-BLK}} = \{0,1,\_\}$.
iv) $\mathsf{ITR} = (\mathsf{ITR.Setup}, \mathsf{ITR.Setup\text{-}Enforce}, \mathsf{ITR.Iterate})$: A cryptographic iterator with an appropriate message space $\mathcal{M}_{\text{ITR}}$.
v) $\mathsf{SPS} = (\mathsf{SPS.Setup}, \mathsf{SPS.Sign}, \mathsf{SPS.Verify}, \mathsf{SPS.Split}, \mathsf{SPS.Sign\text{-}ABO})$: A splittable signature scheme with an appropriate message space $\mathcal{M}_{\text{SPS}}$.
vi) $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$: A length-doubling pseudorandom generator.
vii) $\mathcal{F} = (\mathcal{F}.\mathsf{Setup}, \mathcal{F}.\mathsf{Puncture}, \mathcal{F}.\mathsf{Eval})$: A puncturable pseudorandom function whose domain and range are chosen appropriately. For simplicity, we assume that $\mathcal{F}$ has inputs and outputs of bounded length instead of fixed length inputs and outputs. This assumption can be easily removed by using different PPRF's for different input and output lengths.

Our CPRF construction is described below:

$\mathsf{CPRF.Setup}(1^\lambda) \to \text{SK}_{\text{CPRF}} = (K, \text{HK})$: The setup authority takes as input the security parameter 1^λ and proceeds as follows:
1. It first chooses a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next it generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$.
3. It sets the master CPRF key as $\text{SK}_{\text{CPRF}} = (K, \text{HK})$.

$\mathsf{CPRF.Eval}(\text{SK}_{\text{CPRF}}, x) \to y = \mathcal{F}(K, (h, \ell_x))$: Taking as input the master CPRF key $\text{SK}_{\text{CPRF}} = (K, \text{HK})$ along with an input $x = x_0 \ldots x_{\ell_x - 1} \in \mathcal{X}_{\text{CPRF}}$, where $|x| = \ell_x$, the setup authority executes the following steps:
1. It computes $h = \mathcal{H}_{\text{HK}}(x)$.
2. It outputs the CPRF value on input x to be $y = \mathcal{F}(K, (h, \ell_x))$.

$\mathsf{CPRF.Constrain}(\text{SK}_{\text{CPRF}}, M) \to \text{SK}_{\text{CPRF}}\{M\} = (\text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{CPRF}})$: On input the master CPRF key $\text{SK}_{\text{CPRF}} = (K, \text{HK})$ and a TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle \in \mathbb{M}_\lambda$, the setup authority performs the following steps:
1. At first, it selects PPRF keys $K_1, \ldots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},E} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next, it generates $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
3. Then, it constructs the following obfuscated programs:
   - $\mathcal{P}_1 = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0, w_0, v_0, K_{\text{SPS},E}])$,
   - $\mathcal{P}_2 = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}, \text{PP}_{\text{ITR}}, K_{\text{SPS},E}])$,
   - $\mathcal{P}_3 = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}, K_{\text{SPS},E}])$,
   - $\mathcal{P}_{\text{CPRF}} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}[M, T = 2^\lambda, \text{PP}_{\text{ACC}}, \text{PP}_{\text{ITR}}, K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}])$,

   where programs $\mathsf{Init\text{-}SPS.Prog}$, $\mathsf{Accumulate.Prog}$, $\mathsf{Change\text{-}SPS.Prog}$, and $\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}$ are depicted respectively in Figs. 3.1, 3.2, 3.3 and 3.4.
4. It Provides the constrained key $\text{SK}_{\text{CPRF}}\{M\} = (\text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{CPRF}}) \in \mathcal{K}_{\text{CPRF-CONST}}$ to a legitimate user.

$\mathsf{CPRF.Eval\text{-}Constrained}(\text{SK}_{\text{CPRF}}\{M\}, x) \to y = \mathcal{F}(K, (h, \ell_x))$ or $\perp$: A user takes as input its constrained key $\text{SK}_{\text{CPRF}}\{M\} = (\text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{CPRF}}) \in \mathcal{K}_{\text{CPRF-CONST}}$ corresponding to some legitimate TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$ and an input $x = x_0 \ldots x_{\ell_x - 1} \in \mathcal{X}_{\text{CPRF}}$ with $|x| = \ell_x$. It proceeds as follows:

Constants: Initial TM state q_0, Accumulator value w_0, Iterator value v_0, PPRF key $K_{\text{SPS},E}$

Input: SSB hash value h

Output: Signature $\sigma_{\text{SPS,OUT}}$

1. Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, 0))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
2. Output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},E}, (v_0, q_0, w_0, 0))$.

Fig. 3.1. Init-SPS.Prog

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF key $K_{\text{SPS},E}$

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4.(a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
(b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. 3.2. Accumulate.Prog

Constants: PPRF keys $K_{\text{SPS},A}, K_{\text{SPS},E}$

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Set $m = (v, \text{ST}, w, 0)$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
2.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
(b) Output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. 3.3. Change-SPS.Prog

1. It first computes $h = \mathcal{H}_{\text{HK}}(x)$.
2. Next, it computes $\breve{\sigma}_{\text{SPS},0} = \mathcal{P}_1(h)$.
3. Then for $j = 1, \ldots, \ell_x$, it iteratively performs the following:
   (a) It computes $\pi_{\text{SSB},j-1} \xleftarrow{\$} \mathsf{SSB.Open}(\text{HK}, x, j-1)$.
   (b) It computes $\text{AUX}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1)$.
   (c) It computes $\text{OUT} = \mathcal{P}_2(j-1, x_{j-1}, q_0, w_{j-1}, \text{AUX}_j, v_{j-1}, \breve{\sigma}_{\text{SPS},j-1}, h, \pi_{\text{SSB},j-1})$.
   (d) If $\text{OUT} = \perp$, it outputs OUT. Else, it parses OUT as $\text{OUT} = (w_j, v_j, \breve{\sigma}_{\text{SPS},j})$.
   (e) It computes $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1, x_{j-1})$.
4. It computes $\sigma_{\text{SPS},0} = \mathcal{P}_3(q_0, w_{\ell_x}, v_{\ell_x}, h, \ell_x, \breve{\sigma}_{\text{SPS},\ell_x})$.
5. It sets $\text{POS}_{M,0} = 0$ and $\text{SEED}_0 = \epsilon$.
6. Suppose, M runs for t_x steps on input x. For $t = 1, \ldots, t_x$, it iteratively performs the following steps:
   (a) It computes $(\text{SYM}_{M,t-1}, \pi_{\text{ACC},t-1}) = \mathsf{ACC.Prep\text{-}Read}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1})$.
   (b) It computes $\text{AUX}_{\ell_x+t} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1})$.
   (c) It computes $\text{OUT} = \mathcal{P}_{\text{CPRF}}(t, \text{SEED}_{t-1}, \text{POS}_{M,t-1}, \text{SYM}_{M,t-1}, \text{ST}_{M,t-1}, w_{\ell_x+t-1}, \pi_{\text{ACC},t-1}, \text{AUX}_{\ell_x+t}, v_{\ell_x+t-1}, h, \ell_x, \sigma_{\text{SPS},t-1})$.

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}$

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}}))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   (b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, output $\mathcal{F}(K, (h, \ell_{\text{INP}}))$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. 3.4. $\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}$

(d) If $t = t_x$, it outputs OUT. Otherwise, it parses OUT as $\text{OUT} = (\text{POS}_{M,t}, \text{SYM}_{M,t}^{(\text{WRITE})}, \text{ST}_{M,t}, w_{\ell_x+t}, v_{\ell_x+t}, \sigma_{\text{SPS},t}, \text{SEED}_t)$.
(e) It computes $\text{STORE}_{\ell_x+t} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1}, \text{SYM}_{M,t}^{(\text{WRITE})})$.

Security

Theorem 3.1 (Security of the CPRF Construction of Section 3.4). *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a secure positional accumulator according to Definition 2.4,* ITR *is a secure cryptographic iterator as per Definition 2.5,* SPS *is a secure splittable signature scheme as defined in Definition 2.6, and* PRG *is a secure injective pseudorandom generator,the* CPRF *construction of Section 3.4 satisfies correctness under constraining and selective pseudorandomness as defined in Definition 3.1.*

The proof of Theorem 3.1 is provided in Appendix A.

4 Our Constrained Verifiable Pseudorandom Function for Turing Machines

4.1 Notion

We extend the notion of constrained pseudorandom functions for Turing machines to constrained verifiable pseudorandom function (CVPRF) for Turing machines in the same spirit as it has been extended in case of circuits by Fuchsbauer et al. [Fuc14] and Chandran et al. [CRV14]. As noted earlier, we involve TM's as opposed to circuits in order to accommodate unbounded inputs. Roughly, in case of a (CVPRF), in addition to generating a master evaluation key, the setup

authority also publishes a public verification key. Given, a constrained key associated with a TM and an input accepted by that TM, a user computes the PRF value along with a proof of the fact that the computed PRF value is correct, which can be verified with respect to the public verification key. The formal definition follows:

Definition 4.1 (Constrained Verifiable Pseudorandom Function for Turing Machines: CVPRF). Let $\mathbb{M}_\lambda$ be a family of TM's with (worst-case) running time bounded by $T = 2^\lambda$. A constrained verifiable pseudorandom function (CVPRF) for $\mathbb{M}_\lambda$ with key space $\mathcal{K}_{\text{CVPRF}}$, input domain $\mathcal{X}_{\text{CVPRF}} \subset \{0,1\}^*$, and output space $\mathcal{Y}_{\text{CVPRF}} \subset \{0,1\}^*$ consists of a constrained key space $\mathcal{K}_{\text{CVPRF-CONST}}$, a proof space Π_{CVPRF}, along with the PPT algorithms (CVPRF.Setup, CVPRF.Eval, CVPRF.Prove, CVPRF.Constrain, CVPRF.Prove-Constrained, CVPRF.Verify) which are described below:

- $\mathsf{CVPRF.Setup}(1^\lambda) \to (\text{SK}_{\text{CVPRF}}, \text{VK}_{\text{CVPRF}})$: The setup authority takes as input the security parameter 1^λ and generates a master CVPRF key SK_{CVPRF} along with a public verification key VK_{CVPRF}.

- $\mathsf{CVPRF.Eval}(\text{SK}_{\text{CVPRF}}, x) \to y$: Taking as input the master CVPRF key SK_{CVPRF} and an input $x \in \mathcal{X}_{\text{CVPRF}}$, the trusted authority outputs the value of the function $y \in \mathcal{Y}_{\text{CVPRF}}$. For simplicity of notation, we will denote by $\mathsf{CVPRF}(\text{SK}_{\text{CVPRF}}, x)$ the output of this algorithm.

- $\mathsf{CVPRF.Prove}(\text{SK}_{\text{CVPRF}}, x) \to \pi_{\text{CVPRF}}$: Taking as input the master CVPRF key SK_{CVPRF} and an input $x \in \mathcal{X}_{\text{CVPRF}}$, the trusted authority outputs a proof $\pi_{\text{CVPRF}} \in \Pi_{\text{CVPRF}}$.

- $\mathsf{CVPRF.Constrain}(\text{SK}_{\text{CVPRF}}, M) \to \text{SK}_{\text{CVPRF}}\{M\}$: On input the master CVPRF key SK_{CVPRF} and a TM $M \in \mathbb{M}_\lambda$, the setup authority provides a constrained key $\text{SK}_{\text{CVPRF}}\{M\}$ to a legitimate user.

- $\mathsf{CVPRF.Prove\text{-}Constrained}(\text{SK}_{\text{CVPRF}}\{M\}, x) \to (y, \pi_{\text{CVPRF}})$ or $\perp$: A user takes as input its constrained key $\text{SK}_{\text{CVPRF}}\{M\}$ corresponding to a legitimate TM $M \in \mathbb{M}_\lambda$ and an input $x \in \mathcal{X}_{\text{CVPRF}}$. It outputs either a value-proof pair $(y, \pi_{\text{CVPRF}}) \in \mathcal{Y}_{\text{CVPRF}} \times \Pi_{\text{CVPRF}}$ or $(\perp, \perp)$ indicating failure.

- $\mathsf{CVPRF.Verify}(\text{VK}_{\text{CVPRF}}, x, y, \pi_{\text{CVPRF}}) \to \hat{\beta} \in \{0,1\}$: A verifier takes as input the public verification key VK_{CVPRF}, an input $x \in \mathcal{X}_{\text{CVPRF}}$, a value $y \in \mathcal{Y}_{\text{CVPRF}}$, together with a proof $\pi_{\text{CVPRF}} \in \Pi_{\text{CVPRF}}$. It outputs a bit $\hat{beta} \in \{0,1\}$.

The algorithms CVPRF.Setup, CVPRF.Prove, CVPRF.Constrain and CVPRF.Prove-Constrained are randomized, while the other two algorithms are deterministic. The algorithms satisfy the following properties:

► **Provability**: For any security parameter λ, $(\text{SK}_{\text{CVPRF}}, \text{VK}_{\text{CVPRF}}) \xleftarrow{\$} \mathsf{CVPRF.Setup}(1^\lambda)$, $M \in \mathbb{M}_\lambda$, $\text{SK}_{\text{CVPRF}}\{M\} \xleftarrow{\$} \mathsf{CVPRF.Constrain}(\text{SK}_{\text{CVPRF}}, M)$, $x \in \mathcal{X}_{\text{CVPRF}}$, and $(y, \pi_{\text{CVPRF}}) \xleftarrow{\$} \mathsf{CVPRF.Prove\text{-}Constrained}(\text{SK}_{\text{CVPRF}}\{M\}, x)$, the following holds:

- If $M(x) = 1$, then $y = \mathsf{CVPRF}(\text{SK}_{\text{CVPRF}}, x)$ and $\mathsf{CVPRF.Verify}(\text{VK}_{\text{CVPRF}}, x, y, \pi_{\text{CVPRF}}) = 1$.
- If $M(x) = 0$, then $(y, \pi_{\text{CVPRF}}) = (\perp, \perp)$.

► **Uniqueness**: For any security parameter λ, VK_{CVPRF}, $x \in \mathcal{X}_{\text{CVPRF}}$, $y_0, y_1 \in \mathcal{Y}_{\text{CVPRF}}$, and $\pi_{\text{CVPRF},0}, \pi_{\text{CVPRF},1} \in \Pi_{\text{CVPRF}}$, one of the following holds:

- $y_0 = y_1$.
- $[\mathsf{CVPRF.Verify}(\text{VK}_{\text{CVPRF}}, x, y_0, \pi_{\text{CVPRF},0}) = 0] \bigvee [\mathsf{CVPRF.Verify}(\text{VK}_{\text{CVPRF}}, x, y_1, \pi_{\text{CVPRF},1}) = 0]$.

► **Constraint Hiding**: For any security parameter λ, $(\text{SK}_{\text{CVPRF}}, \text{VK}_{\text{CVPRF}}) \xleftarrow{\$} \mathsf{CVPRF.Setup}(1^\lambda)$, $M \in \mathbb{M}_\lambda$, $\text{SK}_{\text{CVPRF}}\{M\} \xleftarrow{\$} \mathsf{CVPRF.Constrain}(\text{SK}_{\text{CVPRF}}, M)$, and $x \in \mathcal{X}_{\text{CVPRF}}$, the second output

π_{CVPRF} of $\mathsf{CVPRF.Prove\text{-}Constrained}(\text{SK}_{\text{CVPRF}}\{M\}, x)$ and the output of $\mathsf{CVPRF.Prove}(\text{SK}_{\text{CVPRF}}, x)$ are distributed identically.

► **Selective Pseudorandomness**: This property of a CVPRF is defined through the following experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$:

- $\mathcal{A}$ submits a challenge input $x^* \in \mathcal{X}_{\text{CVPRF}}$ to $\mathcal{B}$.
- $\mathcal{B}$ generates $(\text{SK}_{\text{CVPRF}}, \text{VK}_{\text{CVPRF}}) \xleftarrow{\$} \mathsf{CVPRF.Setup}(1^\lambda)$. Next it selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, it computes $y^* = \mathsf{CVPRF}(\text{SK}_{\text{CVPRF}}, x^*)$. Otherwise, it chooses a random $y^* \xleftarrow{\$} \mathcal{Y}_{\text{CVPRF}}$. It returns $(\text{VK}_{\text{CVPRF}}, y^*)$ to $\mathcal{A}$.
- $\mathcal{A}$ may adaptively make a polynomial number of queries of the following kinds to $\mathcal{B}$:
  - **Proof query**: $\mathcal{A}$ queries the CVPRF value along with a proof at some input $x \in \mathcal{X}_{\text{CVPRF}}$ such that $x \neq x^*$. $\mathcal{B}$ provides $(\mathsf{CVPRF}(\text{SK}_{\text{CVPRF}}, x), \mathsf{CVPRF.Prove}(\text{SK}_{\text{CVPRF}}, x))$ to $\mathcal{A}$.
  - **Key query**: $\mathcal{A}$ queries a constrained key corresponding to TM $M \in \mathbb{M}_\lambda$ subject to the constraint that $M(x^*) = 0$. $\mathcal{B}$ gives the constrained key $\text{SK}_{\text{CVPRF}}\{M\} \xleftarrow{\$} \mathsf{CVPRF.Constrain}(\text{SK}_{\text{CVPRF}}, M)$ to $\mathcal{A}$.
- $\mathcal{A}$ eventually outputs a guess bit $b' \in \{0,1\}$.

The CVPRF is said to be selectively pseudorandom if for any PPT adversary $\mathcal{A}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{A}}^{\text{CVPRF,SEL-PR}}(\lambda) = |\mathsf{Pr}[b = b'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

Remark 4.1. Note that following the arguments given in Remark 3.1, we may assume without loss of generality, that the adversary $\mathcal{A}$ in the above selective pseudorandomness experiment only makes constrained key queries and no evaluation query.

4.2 Construction

Here we will provide our CVPRF for TM's. This construction is obtained by extending our CPRF construction described in Section 3.4. Let λ be the underlying security parameter. Let $\mathbb{M}_\lambda$ be a class of TM's, the members of which have (worst-case) running time bounded by $T = 2^\lambda$, input alphabet $\Sigma_{\text{INP}} = \{0,1\}$, and tape alphabet $\Sigma_{\text{TAPE}} = \{0,1,\_\}$. Our CVPRF construction for TM family $\mathbb{M}_\lambda$ will employ all the building blocks utilized in our CPRF construction. Additionally, we will use a perfectly correct and chosen plaintext attack (CPA) secure public key encryption scheme $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Encrypt}, \mathsf{PKE.Decrypt})$ with an appropriate message space. The formal description of our CVPRF construction follows:

$\mathsf{CVPRF.Setup}(1^\lambda) \rightarrow (\text{SK}_{\text{CVPRF}} = (K, K_{\text{PKE}}, \text{HK}), \text{VK}_{\text{CVPRF}} = (\text{HK}, \mathcal{V}_{\text{CVPRF}}))$: The setup authority takes as input the security parameter 1^λ and proceeds as follows:

1. It first chooses PPRF keys $K, K_{\text{PKE}} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next it generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$.
3. Then, it creates the obfuscated program $\mathcal{V}_{\text{CVPRF}} = \mathcal{IO}(\mathsf{Verify.Prog}_{\text{CVPRF}}[K, K_{\text{PKE}}])$, where the program $\mathsf{Verify.Prog}_{\text{CVPRF}}$ is described in Fig. 4.1.
4. It sets the master CVPRF key as $\text{SK}_{\text{CVPRF}} = (K, K_{\text{PKE}}, \text{HK})$ and publishes the public verification key $\text{VK}_{\text{CVPRF}} = (\text{HK}, \mathcal{V}_{\text{CVPRF}})$.

$\mathsf{CVPRF.Eval}(\text{SK}_{\text{CVPRF}}, x) \rightarrow y = \mathcal{F}(K, (h, \ell_x))$: Taking as input the master CVPRF key $\text{SK}_{\text{CVPRF}} = (K, K_{\text{PKE}}, \text{HK})$ along with an input $x = x_0 \ldots x_{\ell_x - 1} \in \mathcal{X}_{\text{CVPRF}}$, where $|x| = \ell_x$, the setup authority executes the following steps:

1. It computes $h = \mathcal{H}_{\text{HK}}(x)$.

Constants: PPRF keys K, K_{PKE}
Inputs: SSB hash value h, Length ℓ_{INP}
Output: (PKE public key $\widehat{\text{PK}}_{\text{PKE}}$, Encryption of CVPRF value $\widehat{\text{CT}}_{\text{PKE}}$)

1. Compute $\hat{r}_{\text{PKE},1} \| \hat{r}_{\text{PKE},2} = \mathcal{F}(K_{\text{PKE}}, (h, \ell_{\text{INP}}))$, $(\widehat{\text{PK}}_{\text{PKE}}, \widehat{\text{SK}}_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda;\ \hat{r}_{\text{PKE},1})$.
2. Compute $\widehat{\text{CT}}_{\text{PKE}} = \mathsf{PKE.Encrypt}(\widehat{\text{PK}}_{\text{PKE}}, \mathcal{F}(K, (h, \ell_{\text{INP}}));\ \hat{r}_{\text{PKE},2})$.
3. Output $(\widehat{\text{PK}}_{\text{PKE}}, \widehat{\text{CT}}_{\text{PKE}})$.

Fig. 4.1. $\mathsf{Verify.Prog}_{\text{CVPRF}}$

2. It outputs the CVPRF value on input x to be $y = \mathcal{F}(K, (h, \ell_x))$.

$\mathsf{CVPRF.Prove}(\text{SK}_{\text{CVPRF}}, x) \rightarrow \pi_{\text{CVPRF}} = (\text{PK}_{\text{PKE}}, r_{\text{PKE},2})$: The setup authority takes as input the master CVPRF key $\text{SK}_{\text{CVPRF}} = (K, K_{\text{PKE}}, \text{HK})$ along with an input $x = x_0 \dots x_{\ell_x - 1} \in \mathcal{X}_{\text{CVPRF}}$, where $|x| = \ell_x$. It proceeds as follows:

1. At first, it computes $h = \mathcal{H}_{\text{HK}}(x)$.
2. Then, it computes $r_{\text{PKE},1} \| r_{\text{PKE},2} = \mathcal{F}(K_{\text{PKE}}, (h, \ell_x))$, $(\text{PK}_{\text{PKE}}, \text{SK}_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda;\ r_{\text{PKE},1})$.
3. It outputs $\pi_{\text{CVPRF}} = (\text{PK}_{\text{PKE}}, r_{\text{PKE},2})$.

$\mathsf{CVPRF.Constrain}(\text{SK}_{\text{CVPRF}}, M) \rightarrow \text{SK}_{\text{CVPRF}}\{M\} = (\text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{CVPRF}})$: On input the master CVPRF key $\text{SK}_{\text{CVPRF}} = (K, K_{\text{PKE}}, \text{HK})$ and a TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle \in \mathbb{M}_\lambda$, the setup authority performs the following steps:

1. At first, it selects PPRF keys $K_1, \dots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},E} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next, it generates $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
3. Then, it constructs the obfuscated programs
   - $\mathcal{P}_1 = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0, w_0, v_0, K_{\text{SPS},E}])$,
   - $\mathcal{P}_2 = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}, \text{PP}_{\text{ITR}}, K_{\text{SPS},E}])$,
   - $\mathcal{P}_3 = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}, K_{\text{SPS},E}])$,
   - $\mathcal{P}_{\text{CVPRF}} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CVPRF}}[M, T = 2^\lambda, \text{PP}_{\text{ACC}}, \text{PP}_{\text{ITR}}, K, K_{\text{PKE}}, K_1, \dots, K_\lambda, K_{\text{SPS},A}])$,

   where the programs Init-SPS.Prog, Accumulate.Prog, and Change-SPS.Prog are as depicted respectively in Figs. 3.1, 3.2 and 3.3 in Section 3.4 while program $\mathsf{Constrained\text{-}Key.Prog}_{\text{CVPRF}}$ is described in Fig. 4.2.
4. It provides the constrained key $\text{SK}_{\text{CVPRF}}\{M\} = (\text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{CVPRF}})$ to a legitimate user.

$\mathsf{CVPRF.Prove\text{-}Constrained}(\text{SK}_{\text{CVPRF}}\{M\}, x) \rightarrow (y = \mathcal{F}(K, (h, \ell_x)), \pi_{\text{CVPRF}} = (\text{PK}_{\text{PKE}}, r_{\text{PKE},2}))$ or $\perp$: A user takes as input its constrained key $\text{SK}_{\text{CVPRF}}\{M\} = (\text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{CVPRF}})$ corresponding to some legitimate TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$ and an input $x = x_0 \dots x_{\ell_x - 1} \in \mathcal{X}_{\text{CVPRF}}$ with $|x| = \ell_x$. It proceeds as follows:

1. It first computes $h = \mathcal{H}_{\text{HK}}(x)$.
2. Next, it computes $\breve{\sigma}_{\text{SPS},0} = \mathcal{P}_1(h)$.
3. Then for $j = 1, \dots, \ell_x$, it iteratively performs the following:
   (a) It computes $\pi_{\text{SSB},j-1} \xleftarrow{\$} \mathsf{SSB.Open}(\text{HK}, x, j-1)$.
   (b) It computes $\text{AUX}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1)$.
   (c) It computes $\text{OUT} = \mathcal{P}_2(j-1, x_{j-1}, q_0, w_{j-1}, \text{AUX}_j, v_{j-1}, \breve{\sigma}_{\text{SPS},j-1}, h, \pi_{\text{SSB},j-1})$.
   (d) If $\text{OUT} = \perp$, it outputs OUT. Else, it parses OUT as $\text{OUT} = (w_j, v_j, \breve{\sigma}_{\text{SPS},j})$.
   (e) It computes $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1, x_{j-1})$.
4. It computes $\sigma_{\text{SPS},0} = \mathcal{P}_3(q_0, w_{\ell_x}, v_{\ell_x}, h, \ell_x, \breve{\sigma}_{\text{SPS},\ell_x})$.
5. It sets $\text{POS}_{M,0} = 0$ and $\text{SEED}_0 = \epsilon$.

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_{\text{PKE}}, K_1, \ldots, K_\lambda, K_{\text{SPS},A}$

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: (CVPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, CVPRF proof $\pi_{\text{CVPRF}} = (\text{PK}_{\text{PKE}}, r_{\text{PKE},2})$) or Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}}))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SSB.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   (b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, perform the following:
   (I) Compute $r_{\text{PKE},1} \| r_{\text{PKE},2} = \mathcal{F}(K_{\text{PKE}}, (h, \ell_{\text{INP}}))$, $(\text{PK}_{\text{PKE}}, \text{SK}_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda;\ r_{\text{PKE},1})$.
   (II) Output $(\mathcal{F}(k, (h, \ell_{\text{INP}})), \pi_{\text{CVPRF}} = (\text{PK}_{\text{PKE}}, r_{\text{PKE},2}))$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. 4.2. $\mathsf{Constrained\text{-}Key.Prog}_{\text{CVPRF}}$

6. Suppose, M runs for t_x steps on input x. For $t = 1, \ldots, t_x$, it iteratively performs the following steps:
   (a) It computes $(\text{SYM}_{M,t-1}, \pi_{\text{ACC},t-1}) = \mathsf{ACC.Prep\text{-}Read}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1})$.
   (b) It computes $\text{AUX}_{\ell_x+t} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1})$.
   (c) It computes $\text{OUT} = \mathcal{P}_{\text{CVPRF}}(t, \text{SEED}_{t-1}, \text{POS}_{M,t-1}, \text{SYM}_{M,t-1}, \text{ST}_{M,t-1}, w_{\ell_x+t-1}, \pi_{\text{ACC},t-1}, \text{AUX}_{\ell_x+t}, v_{\ell_x+t-1}, h, \ell_x, \sigma_{\text{SPS},t-1})$.
   (d) If $t = t_x$, it outputs OUT. Otherwise, it parses OUT as $\text{OUT} = (\text{POS}_{M,t}, \text{SYM}_{M,t}^{(\text{WRITE})}, \text{ST}_{M,t}, w_{\ell_x+t}, v_{\ell_x+t}, \sigma_{\text{SPS},t}, \text{SEED}_t)$.
   (e) It computes $\text{STORE}_{\ell_x+t} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1}, \text{SYM}_{M,t}^{(\text{WRITE})})$.

$\mathsf{CVPRF.Verify}(\text{VK}_{\text{CVPRF}}, x, y, \pi_{\text{CVPRF}}) \rightarrow \hat{\beta} \in \{0, 1\}$: A verifier takes as input the public verification key $\text{VK}_{\text{CVPRF}} = (\text{HK}, \mathcal{V}_{\text{CVPRF}})$, an input $x = x_0 \ldots x_{\ell_x - 1} \in \mathcal{X}_{\text{CVPRF}}$, where $|x| = \ell_x$, a value $y \in \mathcal{Y}_{\text{CVPRF}}$, and a proof $\pi_{\text{CVPRF}} = (\text{PK}_{\text{PKE}}, r) \in \Pi_{\text{CVPRF}}$. It executes the following:

1. It first computes $h = \mathcal{H}_{\text{HK}}(x)$.
2. Next, it computes $(\widehat{\text{PK}}_{\text{PKE}}, \widehat{\text{CT}}_{\text{PKE}}) = \mathcal{V}_{\text{CVPRF}}(h, \ell_x)$.
3. If $[\text{PK}_{\text{PKE}} = \widehat{\text{PK}}_{\text{PKE}}] \ \wedge \ [\mathsf{PKE.Encrypt}(\text{PK}_{\text{PKE}}, y; r) = \widehat{\text{CT}}_{\text{PKE}}]$, it outputs 1. Otherwise, it outputs 0.

Security

Theorem 4.1 (Security of the CVPRF Construction of Section 4.2). *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a secure positional accumulator as defined in Definition 2.4,* ITR *is a secure cryptographic iterator as per Definition 2.5,* SPS *is a secure splitable signature scheme according to Definition 2.6,* PRG *is a secure injective pseudorandom generator, and* PKE *is a*

perfectly correct CPA *secure public key encryption scheme, the* CVPRF *construction described in Section 4.2 satisfies all the properties of a secure* CVPRF *defined in Definition 4.1 of Section 4.1.*

The proof of Theorem 4.1 is given in Appendix B.

5 Our Delegatable Constrained Pseudorandom Function for Turing Machines

5.1 Notion

We now proceed to define the notion of key delegation in the context of CPRF's for TM's along the same line as it has been introduced by Chandran et al. [CRV14]. Roughly, delegatable constrained pseudorandom functions (DCPRF) are the extension of standard CPRF's in which a user holding a constrained key corresponding to some TM can issue constrained keys with the restriction that the issued constrained keys does not enable the evaluation of the PRF value on any additional input beyond those learnable using its own constrained key. Here again the use of TM's in place of circuits, as in [CRV14], allows us to support inputs of appriory unbounded length. The formal definition of DCPRF is described below. We note that although here we consider only one level of delegation, our definition can naturally be extended to support multiple delegation levels.

Definition 5.1 (Delegatable Constrained Pseudorandom Function for Turing Machines: DCPRF). Let $\mathbb{M}_\lambda$ be a family of TM's with (worst-case) running time bounded by $T = 2^\lambda$. A delegatable constrained pseudorandom function (DCPRF) with key space $\mathcal{K}_{\text{DCPRF}}$, input domain $\mathcal{X}_{\text{DCPRF}} \subset \{0,1\}^*$, and output space $\mathcal{Y}_{\text{DCPRF}} \subset \{0,1\}^*$ for the TM family $\mathbb{M}_\lambda$ consists of an additional key space $\mathcal{K}_{\text{DCPRF-CONST}}$ and PPT algorithms (DCPRF.Setup, DCPRF.Eval, DCPRF.Constrain, DCPRF.Delegate, DCPRF.Eval-Constrained) described as follows:

DCPRF.Setup$(1^\lambda) \rightarrow \text{SK}_{\text{DCPRF}}$: The setup authority takes as input the security parameter 1^λ and generates the master DCPRF key $\text{SK}_{\text{DCPRF}} \in \mathcal{K}_{\text{DCPRF}}$.

DCPRF.Eval$(\text{SK}_{\text{DCPRF}}, x) \rightarrow y$: On input the master DCPRF key SK_{DCPRF} along with an input $x \in \mathcal{X}_{\text{DCPRF}}$, the setup authority computes the value of the DCPRF $y \in \mathcal{Y}_{\text{DCPRF}}$. For simplicity of notation, we will use DCPRF$(\text{SK}_{\text{DCPRF}}, x)$ to indicate the output of this algorithm.

DCPRF.Constrain$(\text{SK}_{\text{DCPRF}}, M) \rightarrow \text{SK}_{\text{DCPRF}}\{M\}$: On input the master DCPRF key $\text{SK}_{\text{DCPRF}} \in \mathcal{K}_{\text{DCPRF}}$ and a TM $M \in \mathbb{M}_\lambda$, the setup authority provides a constrained key $\text{SK}_{\text{DCPRF}}\{M\} \in \mathcal{K}_{\text{DCPRF-CONST}}$ to a legitimate user.

DCPRF.Delegate$(\text{SK}_{\text{DCPRF}}\{M\}, \widetilde{M}) \rightarrow \text{SK}_{\text{DCPRF}}\{M \wedge \widetilde{M}\}$: Taking as input a constrained key $\text{SK}_{\text{DCPRF}}\{M\} \in \mathcal{K}_{\text{DCPRF-CONST}}$ corresponding to a legitimate TM $M \in \mathbb{M}_\lambda$ along with another TM $\widetilde{M} \in \mathbb{M}_\lambda$, a user gives a delegated constrained key $\text{SK}_{\text{DCPRF}}\{M \wedge \widetilde{M}\} \in \mathcal{K}_{\text{DCPRF-CONST}}$ to a legitimate delegate.

DCPRF.Eval-Constrained$(\text{SK}_{\text{DCPRF}}\{M\}/\text{SK}_{\text{DCPRF}}\{M \wedge \widetilde{M}\}, x) \rightarrow y$ or $\perp$: A user takes as input a constrained key $\text{SK}_{\text{DCPRF}}\{M\} \in \mathcal{K}_{\text{DCPRF-CONST}}$ obtained from the setup authority, corresponding to TM $M \in \mathbb{M}_\lambda$, or a delegated constrained key $\text{SK}_{\text{DCPRF}}\{M \wedge \widetilde{M}\} \in \mathcal{K}_{\text{DCPRF-CONST}}$ delegated by a constrained key holder holding the constrained key $\text{SK}_{\text{DCPRF}}\{M\} \in \mathcal{K}_{\text{DCPRF-CONST}}$, corresponding to TM $\widetilde{M} \in \mathbb{M}_\lambda$, along with an input $x \in \mathcal{X}_{\text{DCPRF}}$. It outputs either a value $y \in \mathcal{Y}_{\text{DCPRF}}$ or $\perp$ indicating failure.

The algorithms DCPRF.Eval and DCPRF.Eval-Constrained are deterministic, while, all the others are randomized. The algorithms satisfy the following properties:

► **Correctness under Constraining/Delegation**: Let us consider any security parameter λ, $x \in \mathcal{X}_{\text{DCPRF}}, \text{SK}_{\text{DCPRF}} \xleftarrow{\$} \mathsf{DCPRF.Setup}(1^\lambda), M, \widetilde{M} \in \mathbb{M}_\lambda, \text{SK}_{\text{DCPRF}}\{M\} \xleftarrow{\$} \mathsf{DCPRF.Constrain}(\text{SK}_{\text{DCPRF}}, M)$ and $\text{SK}_{\text{DCPRF}}\{M \ \wedge \ \widetilde{M}\} \xleftarrow{\$} \mathsf{DCPRF.Delegate}(\text{SK}_{\text{DCPRF}}\{M\}, \widetilde{M})$. The following must hold:

$$\mathsf{DCPRF.Eval\text{-}Constrained}(\text{SK}_{\text{DCPRF}}\{M\}/\text{SK}_{\text{DCPRF}}\{M \ \wedge \ \widetilde{M}\}, x) = \begin{cases} \mathsf{DCPRF}(\text{SK}_{\text{DCPRF}}, x), \text{ if } M(x) = 1/[M(x) = 1] \ \wedge \ [\widetilde{M}(x) = 1] \\ \perp, \qquad\qquad\qquad\quad \text{otherwise} \end{cases}$$

► **Selective Pseudorandomness**: This property of a DCPRF is defined through the following experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$:

- $\mathcal{A}$ submits a challenge input $x^* \in \mathcal{X}_{\text{DCPRF}}$ to $\mathcal{B}$.
- $\mathcal{B}$ generates $\text{SK}_{\text{DCPRF}} \xleftarrow{\$} \mathsf{DCPRF.Setup}(1^\lambda)$ and selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathsf{DCPRF}(\text{SK}_{\text{DCPRF}}, x^*)$. Otherwise, $\mathcal{B}$ chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{DCPRF}}$. $\mathcal{B}$ returns y^* to $\mathcal{A}$.
- $\mathcal{A}$ may adaptively make a polynomial number of queries of the following types:
  - **Constrained key query**: $\mathcal{A}$ queries a constrained key corresponding to TM $M \in \mathbb{M}_\lambda$ subject to the constraint that $M(x^*) = 0$. $\mathcal{B}$ hands the constrained key $\text{SK}_{\text{DCPRF}}\{M\} \xleftarrow{\$} \mathsf{DCPRF.Constrain}(\text{SK}_{\text{DCPRF}}, M)$ to $\mathcal{A}$.
  - **Delegated key query**: $\mathcal{A}$ queries a delegated key by sending a pair of TM's $(M, \widetilde{M}) \in \mathbb{M}_\lambda^2$ subject to the constraint that $[M(x^*) = 0] \vee [\widetilde{M}(x^*) = 0]$. $\mathcal{B}$ first checks whether $\text{SK}_{\text{DCPRF}}\{M\}$ has already been generated while answering any previous constrained key or delegated key query, and if so, then it creates $\text{SK}_{\text{DCPRF}}\{M \ \wedge \ \widetilde{M}\} \xleftarrow{\$} \mathsf{DCPRF.Delegate}(\text{SK}_{\text{DCPRF}}\{M\}, \widetilde{M})$. On the other hand, if $\text{SK}_{\text{DCPRF}}\{M\}$ has not yet been generated, then $\mathcal{B}$ forms $\text{SK}_{\text{DCPRF}}\{M \ \wedge \ \widetilde{M}\} \xleftarrow{\$} \mathsf{DCPRF.Delegate}(\mathsf{DCPRF.Constrain}(\text{SK}_{\text{DCPRF}}, M), \widetilde{M})$. $\mathcal{B}$ gives the delegated key $\text{SK}_{\text{DCPRF}}\{M \ \wedge \ \widetilde{M}\}$ to $\mathcal{A}$.
  - **Evaluation query**: $\mathcal{A}$ queries the DCPRF value at some input $x \in \mathcal{X}_{\text{DCPRF}}$ such that $x \neq x^*$. $\mathcal{B}$ returns $\mathsf{DCPRF}(\text{SK}_{\text{DCPRF}}, x)$ to $\mathcal{A}$.
- At the end of interaction $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$.

The DCPRF is said to be selectively pseudorandom if for any PPT adversary $\mathcal{A}$, for any security parameter λ,

$$\mathsf{Adv}_{\mathcal{A}}^{\text{DCPRF,SEL-PR}}(\lambda) = |\mathsf{Pr}[b = b'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

Remark 5.1. Note that by a similar reasoning as in Remark 3.1, in the above experiment we can, without loss of generality, replace an evaluation query for some input $x \neq x^* \in \mathcal{X}_{\text{DCPRF}}$ with a constrained key query corresponding to a TM $M_x \in \mathbb{M}_\lambda$ such that M_x accepts only x. Also, note that without loss of generality, we may assume that for any delegated key query of $\mathcal{A}$ corresponding to TM pair $(M, \widetilde{M}) \in \mathbb{M}_\lambda^2$, it holds that $[M(x^*) = 1] \ \wedge \ [\widetilde{M}(x^*) = 0]$ as any delegated key query corresponding to TM pair $(M, \widetilde{M}) \in \mathbb{M}_\lambda^2$ with $M(x^*) = 0$ can be replaced with a valid constrained key query for TM M in view of the fact that once $\mathcal{A}$ posses a constrained key $\text{SK}_{\text{DCPRF}}\{M\}$ it can generate the delegated key $\text{SK}_{\text{DCPRF}}\{M \ \wedge \ \widetilde{M}\}$ for any $\widetilde{M} \in \mathbb{M}_\lambda$ on its own. We will use these simplifications in our proof.

5.2 Construction

In this section, we will present our DCPRF for TM's. The construction presented here considers only one level of delegation, however, it can readily be generalized to support multiple delegation

levels. Let λ be the underlying security parameter. Consider the class $\mathbb{M}_\lambda$ of TM's, the members of which have (worst-case) running time bounded by $T = 2^\lambda$, input alphabet $\Sigma_{\text{INP}} = \{0,1\}$, and tape alphabet $\Sigma_{\text{TAPE}} = \{0,1,\_\}$. Our DCPRF construction is an augmentation of our CPRF construction with a delegation functionality and employs all the cryptographic building blocks utilized by our CPRF construction. In addition, we use a perfectly correct and CPA secure public key encryption scheme $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Encrypt}, \mathsf{PKE.Decrypt})$ with an appropriate message space. The formal description of our DCPRF follows:

$\mathsf{DCPRF.Setup}(1^\lambda) \to \text{SK}_{\text{DCPRF}} = (K, \text{HK})$: The setup authority takes as input the security parameter 1^λ and proceeds as follows:

1. It first chooses a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next it generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$.
3. It sets the master DCPRF key as $\text{SK}_{\text{DCPRF}} = (K, \text{HK})$.

$\mathsf{DCPRF.Eval}(\text{SK}_{\text{DCPRF}}, x) \to y = \mathcal{F}(K, (h, \ell_x))$: Taking as input the master DCPRF key $\text{SK}_{\text{DCPRF}} = (K, \text{HK})$ along with an input $x = x_0 \dots x_{\ell_x - 1} \in \mathcal{X}_{\text{DCPRF}}$, where $|x| = \ell_x$, the setup authority executes the following steps:

1. It computes $h = \mathcal{H}_{\text{HK}}(x)$.
2. It outputs the DCPRF value on input x to be $y = \mathcal{F}(K, (h, \ell_x))$.

$\mathsf{DCPRF.Constrain}(\text{SK}_{\text{DCPRF}}, M) \to \text{SK}_{\text{DCPRF}}\{M\} = (K', \text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{DCPRF}})$: On input the master DCPRF key $\text{SK}_{\text{DCPRF}} = (K, \text{HK})$ and a TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle \in \mathbb{M}_\lambda$, the setup authority performs the following steps:

1. At first, it selects PPRF keys $K', K_1, \dots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},E} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next, it generates $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
3. Then, it constructs the obfuscated programs
   - $\mathcal{P}_1 = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0, w_0, v_0, K_{\text{SPS},E}])$,
   - $\mathcal{P}_2 = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}, \text{PP}_{\text{ITR}}, K_{\text{SPS},E}])$,
   - $\mathcal{P}_3 = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}, K_{\text{SPS},E}])$,
   - $\mathcal{P}_{\text{DCPRF}} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{DCPRF}}[M, T = 2^\lambda, \text{PP}_{\text{ACC}}, \text{PP}_{\text{ITR}}, K, K', K_1, \dots, K_\lambda, K_{\text{SPS},A}])$,

   where the programs Init-SPS.Prog, Accumulate.Prog, and Change-SPS.Prog are depicted respectively in Figs. 3.1, 3.2 and 3.3 in Section 3.4, while the program $\mathsf{Constrained\text{-}Key.Prog}_{\text{DCPRF}}$ is described in Fig. 5.1.
4. It provides the constrained key $\text{SK}_{\text{DCPRF}}\{M\} = (K', \text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{DCPRF}})$ to a legitimate user.

$\mathsf{DCPRF.Delegate}(\text{SK}_{\text{DCPRF}}\{M\}, \widetilde{M}) \to \text{SK}_{\text{DCPRF}}\{M \wedge \widetilde{M}\} = (\widetilde{K}', \text{HK}, \text{PP}_{\text{ACC}}, \widetilde{\text{PP}}_{\text{ACC}}, w_0, \widetilde{w}_0, \text{STORE}_0, \widetilde{\text{STORE}}_0, \text{PP}_{\text{ITR}}, \widetilde{\text{PP}}_{\text{ITR}}, v_0, \widetilde{v}_0, \mathcal{P}_1, \widetilde{\mathcal{P}}_1, \mathcal{P}_2, \widetilde{\mathcal{P}}_2, \mathcal{P}_3, \widetilde{\mathcal{P}}_3, \mathcal{P}_{\text{DCPRF}}, \widetilde{\mathcal{P}}_{\text{DCPRF}})$: A user takes as input a constrained key $\text{SK}_{\text{DCPRF}}\{M\} = (K', \text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{DCPRF}})$, corresponding to a legitimate TM $M \in \mathbb{M}_\lambda$ and another TM $\widetilde{M} = \langle \widetilde{Q}, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \widetilde{\delta}, \widetilde{q}_0, \widetilde{q}_{\text{AC}}, \widetilde{q}_{\text{REJ}} \rangle \in \mathbb{M}_\lambda$. It proceeds as follows:

1. It first picks fresh PPRF keys $\widetilde{K}', \widetilde{K}_1, \dots, \widetilde{K}_\lambda, \widetilde{K}_{\text{SPS},A}, \widetilde{K}_{\text{SPS},E} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next it generates $(\widetilde{\text{PP}}_{\text{ACC}}, \widetilde{w}_0, \widetilde{\text{STORE}}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\widetilde{\text{PP}}_{\text{ITR}}, \widetilde{v}_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$ afresh.
3. Then, it constructs the obfuscated programs
   - $\widetilde{\mathcal{P}}_1 = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[\widetilde{q}_0, \widetilde{w}_0, \widetilde{v}_0, \widetilde{K}_{\text{SPS},E}])$,
   - $\widetilde{\mathcal{P}}_2 = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \widetilde{\text{PP}}_{\text{ACC}}, \widetilde{\text{PP}}_{\text{ITR}}, \widetilde{K}_{\text{SPS},E}])$,
   - $\widetilde{\mathcal{P}}_3 = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[\widetilde{K}_{\text{SPS},A}, \widetilde{K}_{\text{SPS},E}])$,

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K', K_1, \ldots, K_\lambda, K_{\text{SPS},A}$

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Encryption of DCPRF value CT_{PKE}, or Header Position (POS_{OUT}, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   (b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, perform the following steps:
   (I) Compute $r_{\text{PKE},1} \| r_{\text{PKE},2} = \mathcal{F}(K', (h, \ell_{\text{INP}}))$, $(\text{PK}_{\text{PKE}}, \text{SK}_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda;\ r_{\text{PKE},1})$.
   (II) Output $\text{CT}_{\text{PKE}} = \mathsf{PKE.Encrypt}(\text{PK}_{\text{PKE}}, \mathcal{F}(K, (h, \ell_{\text{INP}}));\ r_{\text{PKE},2})$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. 5.1. $\mathsf{Constrained\text{-}Key.Prog}_{\text{DCPRF}}$

– $\widetilde{\mathcal{P}}_{\text{DCPRF}} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{DCPRF}}[\widetilde{M}, T = 2^\lambda, \widetilde{\text{PP}}_{\text{ACC}}, \widetilde{\text{PP}}_{\text{ITR}}, K', \widetilde{K'}, \widetilde{K}_1, \ldots, \widetilde{K}_\lambda, \widetilde{K}_{\text{SPS},A}])$,

where the programs $\mathsf{Init\text{-}SPS.Prog}$, $\mathsf{Accumulate.Prog}$, and $\mathsf{Change\text{-}SPS.Prog}$ are depicted respectively in Figs. 3.1, 3.2 and 3.3 in Section 3.4, while the program $\mathsf{Constrained\text{-}Key.Prog}_{\text{DCPRF}}$ is described in Fig. 5.1.

4. It gives the delegated key $\text{SK}_{\text{DCPRF}}\{M \ \wedge \ \widetilde{M}\} = (\widetilde{K'}, \text{HK}, \text{PP}_{\text{ACC}}, \widetilde{\text{PP}}_{\text{ACC}}, w_0, \widetilde{w}_0, \text{STORE}_0, \widetilde{\text{STORE}}_0, \text{PP}_{\text{ITR}}, \widetilde{\text{PP}}_{\text{ITR}}, v_0, \widetilde{v}_0, \mathcal{P}_1, \widetilde{\mathcal{P}}_1, \mathcal{P}_2, \widetilde{\mathcal{P}}_2, \mathcal{P}_3, \widetilde{\mathcal{P}}_3, \mathcal{P}_{\text{DCPRF}}, \widetilde{\mathcal{P}}_{\text{DCPRF}})$ to a legitimate delegate.

$\mathsf{DCPRF.Eval\text{-}Constrained}(\text{SK}_{\text{DCPRF}}\{M\}/\text{SK}_{\text{DCPRF}}\{M \ \wedge \ \widetilde{M}\}, x) \rightarrow y = \mathcal{F}(K, (h, \ell_x))$ or $\perp$: A user takes as input a constrained key $\text{SK}_{\text{DCPRF}}\{M\} = (K', \text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{DCPRF}})$ obtained from the setup authority, corresponding to some legitimate TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle \in \mathbb{M}_\lambda$, or a delegated key $\text{SK}_{\text{DCPRF}}\{M \ \wedge \ \widetilde{M}\} = (\widetilde{K'}, \text{HK}, \text{PP}_{\text{ACC}}, \widetilde{\text{PP}}_{\text{ACC}}, w_0, \widetilde{w}_0, \text{STORE}_0, \widetilde{\text{STORE}}_0, \text{PP}_{\text{ITR}}, \widetilde{\text{PP}}_{\text{ITR}}, v_0, \widetilde{v}_0, \mathcal{P}_1, \widetilde{\mathcal{P}}_1, \mathcal{P}_2, \widetilde{\mathcal{P}}_2, \mathcal{P}_3, \widetilde{\mathcal{P}}_3, \mathcal{P}_{\text{DCPRF}}, \widetilde{\mathcal{P}}_{\text{DCPRF}})$ obtained from the holder of the constrained key $\text{SK}_{\text{DCPRF}}\{M\}$, corresponding to TM $\widetilde{M} = \langle \widetilde{Q}, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \widetilde{\delta}, \widetilde{q}_0, \widetilde{q}_{\text{AC}}, \widetilde{q}_{\text{REJ}} \rangle \in \mathbb{M}_\lambda$, along with an input $x = x_0 \ldots x_{\ell_x - 1} \in \mathcal{X}_{\text{DCPRF}}$ with $|x| = \ell_x$. It proceeds as follows:

(A) If $M(x) = 0$, it outputs $\perp$. Otherwise, it performs the following steps:
   1. It first computes $h = \mathcal{H}_{\text{HK}}(x)$.
   2. Next, it computes $\breve{\sigma}_{\text{SPS},0} = \mathcal{P}_1(h)$.
   3. Then for $j = 1, \ldots, \ell_x$, it iteratively performs the following:
      (a) It computes $\pi_{\text{SSB},j-1} \xleftarrow{\$} \mathsf{SSB.Open}(\text{HK}, x, j-1)$.
      (b) It computes $\text{AUX}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1)$.
      (c) It computes $\text{OUT} = \mathcal{P}_2(j-1, x_{j-1}, q_0, w_{j-1}, \text{AUX}_j, v_{j-1}, \breve{\sigma}_{\text{SPS},j-1}, h, \pi_{\text{SSB},j-1})$.
      (d) If $\text{OUT} = \perp$, it outputs OUT. Else, it parses OUT as $\text{OUT} = (w_j, v_j, \breve{\sigma}_{\text{SPS},j})$.

(e) It computes $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1, x_{j-1})$.

4. It computes $\sigma_{\text{SPS},0} = \mathcal{P}_3(q_0, w_{\ell_x}, v_{\ell_x}, h, \ell_x, \breve{\sigma}_{\text{SPS},\ell_x})$.
5. It sets $\text{POS}_{M,0} = 0$ and $\text{SEED}_0 = \epsilon$.
6. Suppose, M accepts x in t_x steps. For $t = 1, \ldots, t_x$, it iteratively performs the following steps:
   (a) It computes $(\text{SYM}_{M,t-1}, \pi_{\text{ACC},t-1}) = \mathsf{ACC.Prep\text{-}Read}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1})$.
   (b) It computes $\text{AUX}_{\ell_x+t} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1})$.
   (c) It computes $\text{OUT} = \mathcal{P}_{\text{DCPRF}}(t, \text{SEED}_{t-1}, \text{POS}_{M,t-1}, \text{SYM}_{M,t-1}, \text{ST}_{M,t-1}, w_{\ell_x+t-1}, \pi_{\text{ACC},t-1}, \text{AUX}_{\ell_x+t}, v_{\ell_x+t-1}, h, \ell_x, \sigma_{\text{SPS},t-1})$.
   (d) If $t = t_x$, it sets $\text{CT}_{\text{PKE}} = \text{OUT}$. Otherwise, it parses OUT as $\text{OUT} = (\text{POS}_{M,t}, \text{SYM}_{M,t}^{(\text{WRITE})}, \text{ST}_{M,t}, w_{\ell_x+t}, v_{\ell_x+t}, \sigma_{\text{SPS},t}, \text{SEED}_t)$.
   (e) It computes $\text{STORE}_{\ell_x+t} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1}, \text{SYM}_{M,t}^{(\text{WRITE})})$.

(B) If the user is using the constrained key $\text{SK}_{\text{DCPRF}}\{M\}$, then it computes $r_{\text{PKE},1} \| r_{\text{PKE},2} = \mathcal{F}(K', (h, \ell_x)), (\text{PK}_{\text{PKE}}, \text{SK}_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda;\ r_{\text{PKE},1})$, and outputs $\mathsf{PKE.Decrypt}(\text{SK}_{\text{PKE}}, \text{CT}_{\text{PKE}})$. On the other hand, if the user is using the delegated key $\text{SK}_{\text{DCPRF}}\{M \wedge \widetilde{M}\}$ and $\widetilde{M}(x) = 0$, then it outputs $\perp$, while if $\widetilde{M}(x) = 1$, it further executes the following steps:

1. It computes $\widetilde{\breve{\sigma}}_{\text{SPS},0} = \widetilde{\mathcal{P}}_1(h)$.
2. Then for $j = 1, \ldots, \ell_x$, it iteratively performs the following:
   (a) It computes $\widetilde{\pi}_{\text{SSB},j-1} \xleftarrow{\$} \mathsf{SSB.Open}(\text{HK}, x, j-1)$.
   (b) It computes $\widetilde{\text{AUX}}_j = \mathsf{ACC.Prep\text{-}Write}(\widetilde{\text{PP}}_{\text{ACC}}, \widetilde{\text{STORE}}_{j-1}, j-1)$.
   (c) It computes $\widetilde{\text{OUT}} = \widetilde{\mathcal{P}}_2(j-1, x_{j-1}, \widetilde{q}_0, \widetilde{w}_{j-1}, \widetilde{\text{AUX}}_j, \widetilde{v}_{j-1}, \widetilde{\breve{\sigma}}_{\text{SPS},j-1}, h, \widetilde{\pi}_{\text{SSB},j-1})$.
   (d) If $\widetilde{\text{OUT}} = \perp$, it outputs $\widetilde{\text{OUT}}$. Else, it parses $\widetilde{\text{OUT}}$ as $\widetilde{\text{OUT}} = (\widetilde{w}_j, \widetilde{v}_j, \widetilde{\breve{\sigma}}_{\text{SPS},j})$.
   (e) It computes $\widetilde{\text{STORE}}_j = \mathsf{ACC.Write\text{-}Store}(\widetilde{\text{PP}}_{\text{ACC}}, \widetilde{\text{STORE}}_{j-1}, j-1, x_{j-1})$.
3. It computes $\widetilde{\sigma}_{\text{SPS},0} = \widetilde{\mathcal{P}}_3(\widetilde{q}_0, \widetilde{w}_{\ell_x}, \widetilde{v}_{\ell_x}, h, \ell_x, \widetilde{\breve{\sigma}}_{\text{SPS},\ell_x})$.
4. It sets $\text{POS}_{\widetilde{M},0} = 0$ and $\widetilde{\text{SEED}}_0 = \epsilon$.
5. Suppose, $\widetilde{M}$ accepts x in $\widetilde{t}_x$ steps. For $t = 1, \ldots, \widetilde{t}_x$, it iteratively performs the following steps:
   (a) It computes $(\text{SYM}_{\widetilde{M},t-1}, \widetilde{\pi}_{\text{ACC},t-1}) = \mathsf{ACC.Prep\text{-}Read}(\widetilde{\text{PP}}_{\text{ACC}}, \widetilde{\text{STORE}}_{\ell_x+t-1}, \text{POS}_{\widetilde{M},t-1})$.
   (b) It computes $\widetilde{\text{AUX}}_{\ell_x+t} = \mathsf{ACC.Prep\text{-}Write}(\widetilde{\text{PP}}_{\text{ACC}}, \widetilde{\text{STORE}}_{\ell_x+t-1}, \text{POS}_{\widetilde{M},t-1})$.
   (c) It computes $\widetilde{\text{OUT}} = \widetilde{\mathcal{P}}_{\text{DCPRF}}(t, \widetilde{\text{SEED}}_{t-1}, \text{POS}_{\widetilde{M},t-1}, \text{SYM}_{\widetilde{M},t-1}, \text{ST}_{\widetilde{M},t-1}, \widetilde{w}_{\ell_x+t-1}, \widetilde{\pi}_{\text{ACC},t-1}, \widetilde{\text{AUX}}_{\ell_x+t}, \widetilde{v}_{\ell_x+t-1}, h, \ell_x, \widetilde{\sigma}_{\text{SPS},t-1})$.
   (d) If $t = \widetilde{t}_x$, it sets $\widetilde{\text{CT}}_{\text{PKE}} = \widetilde{\text{OUT}}$. Otherwise, it parses $\widetilde{\text{OUT}}$ as $\widetilde{\text{OUT}} = (\text{POS}_{\widetilde{M},t}, \text{SYM}_{\widetilde{M},t}^{(\text{WRITE})}, \text{ST}_{\widetilde{M},t}, \widetilde{w}_{\ell_x+t}, \widetilde{v}_{\ell_x+t}, \widetilde{\sigma}_{\text{SPS},t}, \widetilde{\text{SEED}}_t)$.
   (e) It computes $\widetilde{\text{STORE}}_{\ell_x+t} = \mathsf{ACC.Write\text{-}Store}(\widetilde{\text{PP}}_{\text{ACC}}, \widetilde{\text{STORE}}_{\ell_x+t-1}, \text{POS}_{\widetilde{M},t-1}, \text{SYM}_{\widetilde{M},t}^{(\text{WRITE})})$.

(C) Finally, it computes
- $\widetilde{r}_{\text{PKE},1} \| \widetilde{r}_{\text{PKE},2} = \mathcal{F}(\widetilde{K'}, (h, \ell_x))$,
- $(\widetilde{\text{PK}}_{\text{PKE}}, \widetilde{\text{SK}}_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda;\ \widetilde{r}_{\text{PKE},1})$,
- $r_{\text{PKE},1} \| r_{\text{PKE},2} = \mathsf{PKE.Decrypt}(\widetilde{\text{SK}}_{\text{PKE}}, \widetilde{\text{CT}}_{\text{PKE}})$,
- $(\text{PK}_{\text{PKE}}, \text{SK}_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda;\ r_{\text{PKE},1})$,

and outputs $\mathsf{PKE.Decrypt}(\text{SK}_{\text{PKE}}, \text{CT}_{\text{PKE}})$.

Security

Theorem 5.1 (Security of the DCPRF Construction of Section 5.2). *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a secure positional accumulator as defined in Definition 2.4,* ITR

is a secure cryptographic iterator as per Definition 2.5, SPS *is a secure splittable signature scheme according to Definition 2.6,* PRG *is a secure injective pseudorandom generator, and* PKE *is* CPA *secure, the* DCPRF *construction of Section 5.2 satisfies the correctness and selective pseudorandomness properties defined in Definition 5.1.*

The proof of Theorem 5.1 is given in Appendix C.

6 Application: Attribute-Based Signature for Turing Machines

6.1 Notion

Here we will formally define the notion of an attribute-based signature scheme where signing policies are associated with TM's. This definition is similar to that defined in [TLL14, SAH16] for circuits. However, due to the use of TM's as opposed to circuits, the scheme can handle signing attribute strings of arbitrary polynomial length.

Definition 6.1 (Attribute-Based Signature for Turing Machines: ABS). Let $\mathbb{M}_\lambda$ be a class of TM's, the members of which have (worst-case) running time bounded by $T = 2^\lambda$. An attribute-based signature (ABS) scheme for signing policies associated with the TM's in $\mathbb{M}_\lambda$ comprises of an attribute universe $\mathcal{U}_{\text{ABS}} \subset \{0,1\}^*$, a message space $\mathcal{M}_{\text{ABS}} = \{0,1\}^{\ell_{\text{ABS-MSG}}}$, a signature space $\mathcal{S}_{\text{ABS}} = \{0,1\}^{\ell_{\text{ABS-SIG}}}$, where $\ell_{\text{ABS-MSG}}, \ell_{\text{ABS-SIG}}$ are some polynomials in the security parameter λ, and PPT algorithms (ABS.Setup, ABS.KeyGen, ABS.Sign, ABS.Verify) described below:

$\mathsf{ABS.Setup}(1^\lambda) \rightarrow (\text{PP}_{\text{ABS}}, \text{MSK}_{\text{ABS}})$: The setup authority takes as input the security parameter 1^λ. It publishes the public parameters PP_{ABS} while generates a master secret key MSK_{ABS} for itself.

$\mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M) \rightarrow \text{SK}_{\text{ABS}}(M)$: Taking as input the master secret key MSK_{ABS} and a signing policy TM $M \in \mathbb{M}_\lambda$ of a signer, the setup authority provides the corresponding signing key $\text{SK}_{\text{ABS}}(M)$ to the legitimate signer.

$\mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M), x, \mathsf{msg}) \rightarrow \sigma_{\text{ABS}}$ or $\perp$: On input the signing key $\text{SK}_{\text{ABS}}(M)$ corresponding to the legitimate signing policy TM $M \in \mathbb{M}_\lambda$, a signing attribute string $x \in \mathcal{U}_{\text{ABS}}$, and a message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$, a signer outputs either a signature $\sigma_{\text{ABS}} \in \mathcal{S}_{\text{ABS}}$ or $\perp$ indicating failure.

$\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x, \mathsf{msg}, \sigma_{\text{ABS}}) \rightarrow \hat{\beta} \in \{0,1\}$: A verifier takes as input the public parameters PP_{ABS}, a signing attribute string $x \in \mathcal{U}_{\text{ABS}}$, a message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$, and a purported signature $\sigma_{\text{ABS}} \in \mathcal{S}_{\text{ABS}}$. It outputs a bit $\hat{\beta} \in \{0,1\}$.

We note that all the algorithms described above except ABS.Verify are randomized. The algorithms satisfy the following properties:

► **Correctness**: For any security parameter λ, $(\text{PP}_{\text{ABS}}, \text{MSK}_{\text{ABS}}) \xleftarrow{\$} \mathsf{ABS.Setup}(1^\lambda)$, $M \in \mathbb{M}_\lambda$, $\text{SK}_{\text{ABS}}(M) \xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M)$, $x \in \mathcal{U}_{\text{ABS}}$, and $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$, if $M(x) = 1$, then $\mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M), x, \mathsf{msg})$ outputs $\sigma_{\text{ABS}} \in \mathcal{S}_{\text{ABS}}$ such that $\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x, \mathsf{msg}, \sigma_{\text{ABS}}) = 1$.

► **Signer Privacy**: An ABS scheme is said to provide signer privacy if for any security parameter λ, message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$, $(\text{PP}_{\text{ABS}}, \text{MSK}_{\text{ABS}}) \xleftarrow{\$} \mathsf{ABS.Setup}(1^\lambda)$, signing policies $M, M' \in \mathbb{M}_\lambda$, signing keys $\text{SK}_{\text{ABS}}(M) \xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M), \text{SK}_{\text{ABS}}(M') \xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M')$, $x \in \mathcal{U}_{\text{ABS}}$ such that $M(x) = 1 = M'(x)$, the distributions of the signatures outputted by $\mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M), x, \mathsf{msg})$ and $\mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M'), x, \mathsf{msg})$ are identical.

► **Existential Unforgeability against Selective Attribute Adaptive Chosen Message Attack**: This property of an ABS scheme is defined through the following experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$:

- $\mathcal{A}$ submits a challenge attribute string $x^* \in \mathcal{U}_{\text{ABS}}$ to $\mathcal{B}$.
- $\mathcal{B}$ generates $(\text{PP}_{\text{ABS}}, \text{MSK}_{\text{ABS}}) \xleftarrow{\$} \mathsf{ABS.Setup}(1^\lambda)$ and provides $\mathcal{A}$ with PP_{ABS}.
- $\mathcal{A}$ may adaptively make a polynomial number of queries of the following types:
  - **Signing key query**: When $\mathcal{A}$ queries a signing key corresponding to a signing policy TM $M \in \mathbb{M}_\lambda$ subject to the restriction that $M(x^*) = 0$, $\mathcal{B}$ gives back $\text{SK}_{\text{ABS}}(M) \xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M)$ to $\mathcal{A}$.
  - **Signature query**: When $\mathcal{A}$ queries a signature on a message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$ under an attribute string $x \in \mathcal{U}_{\text{ABS}}$, $\mathcal{B}$ samples a signing policy TM $M \in \mathbb{M}_\lambda$ such that $M(x) = 1$, creates a signing key $\text{SK}_{\text{ABS}}(M) \xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M)$, and generates a signature $\sigma_{\text{ABS}} \xleftarrow{\$} \mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M), x, \mathsf{msg})$, which $\mathcal{B}$ returns to $\mathcal{A}$.
- At the end of interaction $\mathcal{A}$ outputs a message-signature pair $(\mathsf{msg}^*, \sigma^*_{\text{ABS}})$. $\mathcal{A}$ wins if the following hold simultaneously:
  i) $\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x^*, \mathsf{msg}^*, \sigma^*_{\text{ABS}}) = 1$.
  ii) $\mathcal{A}$ has not made any signature query on msg^* under x^*.

The ABS scheme is said to be existentially unforgeable against selective attribute adaptive chosen message attack if for any PPT adversary $\mathcal{A}$, for any security parameter λ,

$$\mathsf{Adv}^{\text{ABS,UF-CMA}}_{\mathcal{A}}(\lambda) = \mathsf{Pr}[\mathcal{A} \text{ wins}] \leq \mathsf{negl}(\lambda)$$

for some negligible function negl.

Remark 6.1. Note that in the existential unforgeability experiment above without loss of generality, we can consider signature queries on messages only under the challenge attribute string x^*. This is because any signature query under some attribute string $x \neq x^*$ can be replaced by a signing key query for a signing policy TM $M_x \in \mathbb{M}_\lambda$ that accepts only the string x. Since $x \neq x^*$, $M_x(x^*) = 0$, and thus M_x forms a valid signing key query. We will use this simplification in our proof.

6.2 Construction

In this section we will present our ABS scheme for TM's. Let λ be the underlying security parameter. Let $\mathbb{M}_\lambda$ denote a family of TM's, the members of which have (worst case) running time bounded by $T = 2^\lambda$, input alphabet $\Sigma_{\text{INP}} = \{0, 1\}$, and tape alphabet $\Sigma_{\text{TAPE}} = \{0, 1, \_\}$. Our ABS scheme closely resembles our CPRF construction described in Section 3.4 and utilizes the same underlying cryptographic tools. Additionally, here we use a digital signature scheme $\mathsf{SIG} = (\mathsf{SIG.Setup}, \mathsf{SIG.Sign}, \mathsf{SIG.Verify})$ which is existentially unforgeable against chosen message attack (CMA). The formal description of our ABS construction follows:

$\mathsf{ABS.Setup}(1^\lambda) \to (\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{V}_{\text{ABS}}), \text{MSK}_{\text{ABS}} = (K, \text{HK}))$: The setup authority takes as input the security parameter 1^λ and proceeds as follows:

1. It first chooses a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next it generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$.
3. Then, it creates the obfuscated program $\mathcal{V}_{\text{ABS}} = \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K])$, where the program $\mathsf{Verify.Prog}_{\text{ABS}}$ is described in Fig. 6.1.

Constants: PPRF key K
Inputs: SSB hash value h, Length ℓ_{INP}
Output: SIG verification key $\widehat{\text{VK}}_{\text{SIG}}$

1. Compute $\hat{r}_{\text{SIG}} = \mathcal{F}(K, (h, \ell_{\text{INP}}))$, $(\widehat{\text{SK}}_{\text{SIG}}, \widehat{\text{VK}}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda;\ \hat{r}_{\text{SIG}})$.
2. Output $\widehat{\text{VK}}_{\text{SIG}}$.

Fig. 6.1. $\mathsf{Verify.Prog}_{\text{ABS}}$

4. It keeps the master secret key $\text{MSK}_{\text{ABS}} = (K, \text{HK})$ and publishes the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{V}_{\text{ABS}})$.

$\mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M) \rightarrow \text{SK}_{\text{ABS}}(M) = (\text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{ABS}})$: On input the master secret key $\text{MSK}_{\text{ABS}} = (K, \text{HK})$ and a signing policy TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}}\rangle \in \mathbb{M}_\lambda$, the setup authority performs the following steps:

1. At first, it selects PPRF keys $K_1, \ldots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},E} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next, it generates $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
3. Then, it constructs the obfuscated programs
   - $\mathcal{P}_1 = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0, w_0, v_0, K_{\text{SPS},E}])$,
   - $\mathcal{P}_2 = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}, \text{PP}_{\text{ITR}}, K_{\text{SPS},E}])$,
   - $\mathcal{P}_3 = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}, K_{\text{SPS},E}])$,
   - $\mathcal{P}_{\text{ABS}} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}[M, T = 2^\lambda, \text{PP}_{\text{ACC}}, \text{PP}_{\text{ITR}}, K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}])$,

   where the programs Init-SPS.Prog, Accumulate.Prog, and Change-SPS.Prog are as depicted respectively in Figs. 3.1, 3.2 and 3.3 in Section 3.4, while the program $\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}$ is described in Fig. 6.2.
4. It provides the constrained key $\text{SK}_{\text{ABS}}(M) = (\text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{ABS}})$ to a legitimate signer.

$\mathsf{ABS.Sign}(\text{SK}_{\text{ABS}}(M), x, \mathsf{msg}) \rightarrow \sigma_{\text{ABS}} = (\text{VK}_{\text{SIG}}, \sigma_{\text{SIG}})$ or $\perp$: A signer takes as input its signing key $\text{SK}_{\text{ABS}}(M) = (\text{HK}, \text{PP}_{\text{ACC}}, w_0, \text{STORE}_0, \text{PP}_{\text{ITR}}, v_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_{\text{ABS}})$, corresponding to its legitimate signing policy TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}}\rangle \in \mathbb{M}_\lambda$, an attribute string $x = x_0 \ldots x_{\ell_x - 1} \in \mathcal{U}_{\text{ABS}}$ with $|x| = \ell_x$, and a message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$. If $M(x) = 0$, it outputs $\perp$. Otherwise, it proceeds as follows:

1. It first computes $h = \mathcal{H}_{\text{HK}}(x)$.
2. Next, it computes $\breve{\sigma}_{\text{SPS},0} = \mathcal{P}_1(h)$.
3. Then for $j = 1, \ldots, \ell_x$, it iteratively performs the following:
   (a) It computes $\pi_{\text{SSB},j-1} \xleftarrow{\$} \mathsf{SSB.Open}(\text{HK}, x, j-1)$.
   (b) It computes $\text{AUX}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1)$.
   (c) It computes $\text{OUT} = \mathcal{P}_2(j-1, x_{j-1}, q_0, w_{j-1}, \text{AUX}_j, v_{j-1}, \breve{\sigma}_{\text{SPS},j-1}, h, \pi_{\text{SSB},j-1})$.
   (d) If $\text{OUT} = \perp$, it outputs OUT. Else, it parses OUT as $\text{OUT} = (w_j, v_j, \breve{\sigma}_{\text{SPS},j})$.
   (e) It computes $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1, x_{j-1})$.
4. It computes $\sigma_{\text{SPS},0} = \mathcal{P}_3(q_0, w_{\ell_x}, v_{\ell_x}, h, \ell_x, \breve{\sigma}_{\text{SPS},\ell_x})$.
5. It sets $\text{POS}_{M,0} = 0$ and $\text{SEED}_0 = \epsilon$.
6. Suppose, M accepts x in t_x steps. For $t = 1, \ldots, t_x$, it iteratively performs the following steps:
   (a) It computes $(\text{SYM}_{M,t-1}, \pi_{\text{ACC},t-1}) = \mathsf{ACC.Prep\text{-}Read}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1})$.
   (b) It computes $\text{AUX}_{\ell_x+t} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1})$.
   (c) It computes $\text{OUT} = \mathcal{P}_{\text{ABS}}(t, \text{SEED}_{t-1}, \text{POS}_{M,t-1}, \text{SYM}_{M,t-1}, \text{ST}_{M,t-1}, w_{\ell_x+t-1}, \pi_{\text{ACC},t-1}, \text{AUX}_{\ell_x+t}, v_{\ell_x+t-1}, h, \ell_x, \sigma_{\text{SPS},t-1})$.
   (d) If $t = t_x$, it parses OUT as $\text{OUT} = (\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}})$. Otherwise, it parses OUT as $\text{OUT} = (\text{POS}_{M,t}, \text{SYM}_{M,t}^{(\text{WRITE})}, \text{ST}_{M,t}, w_{\ell_x+t}, v_{\ell_x+t}, \sigma_{\text{SPS},t}, \text{SEED}_t)$.

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}$

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: (SIG signing key SK_{SIG}, SIG verification key VK_{SIG}), or Header Position (POS_{OUT}, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SSB.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   (b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, perform the following:
   (I) Compute $r_{\text{SIG}} = \mathcal{F}(K, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda;\ r_{\text{SIG}})$.
   (II) Output $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}})$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. 6.2. $\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}$

(e) It computes $\text{STORE}_{\ell_x+t} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{\ell_x+t-1}, \text{POS}_{M,t-1}, \text{SYM}_{M,t}^{(\text{WRITE})})$.

7. Finally, it computes $\sigma_{\text{SIG}} \xleftarrow{\$} \mathsf{SIG.Sign}(\text{SK}_{\text{SIG}}, \mathsf{msg})$.
8. It outputs the signature $\sigma_{\text{ABS}} = (\text{VK}_{\text{SIG}}, \sigma_{\text{SIG}}) \in \mathcal{S}_{\text{ABS}}$.

$\mathsf{ABS.Verify}(\text{PP}_{\text{ABS}}, x, \mathsf{msg}, \sigma_{\text{ABS}}) \rightarrow \hat{\beta} \in \{0,1\}$: A verifier takes as input the public parameters $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{V}_{\text{ABS}})$, an attribute string $x = x_0 \ldots x_{\ell_x - 1} \in \mathcal{U}_{\text{ABS}}$, where $|x| = \ell_x$, a message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$, together with a signature $\sigma_{\text{ABS}} = (\text{VK}_{\text{SIG}}, \sigma_{\text{SIG}}) \in \mathcal{S}_{\text{ABS}}$. It executes the following:

1. It first computes $h = \mathcal{H}_{\text{HK}}(x)$.
2. Next, it computes $\widehat{\text{VK}}_{\text{SIG}} = \mathcal{V}_{\text{ABS}}(h, \ell_x)$.
3. If $[\text{VK}_{\text{SIG}} = \widehat{\text{VK}}_{\text{SIG}}] \ \wedge \ [\mathsf{SIG.Verify}(\text{VK}_{\text{SIG}}, \mathsf{msg}, \sigma_{\text{SIG}}) = 1]$, it outputs 1. Otherwise, it outputs 0.

Security

Theorem 6.1 (Security of the ABS Scheme of Section 6.2). *Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a secure positional accumulator as defined in Definition 2.4,* ITR *is a secure cryptographic iterator as per Definition 2.5,* SPS *is a secure splittable signature scheme according to Definition 2.6,* PRG *is a secure injective pseudorandom generator, and* SIG *is existentially unforgeable against chosen message attack, the* ABS *scheme of Section 6.2 satisfies all the criteria of a secure* ABS *defined in Definition 6.1.*

The proof of Theorem 6.1 is provided in Appendix D.

7 Conclusion

In this paper, besides fixing the flaw in the security argument of the IO-based CPRF construction of [DKW16] for inputs of unbounded polynomial length, we have presented the *first* constructions of CVPRF and DCPRF supporting *unbounded* length inputs. In view of the countless applications of CPRF's and their various extensions in resolving exciting fundamental problems of modern cryptography, it is desirable to improve the efficiency as well as strengthen the security of these primitives based on well-studied cryptographic tools and complexity assumptions. Although, our results have proved existence of CPRF, CVPRF, and DCPRF for unconstrained length inputs, the constructions are quite expensive given the research progress in the field of IO so far. Therefore, an inportant research direction is to seek for cost-effective constructions of CPRF and its enhancements which can handle inputs of arbitrary polynomial length, without employing such heavy duty tools like IO or multilinear maps. A more fundamental challenge is to investigate adaptively secure constructions of these primitives supporting apriori unbounded length inputs without using the technique of complexity leveraging.

References

[AF16] Hamza Abusalah and Georg Fuchsbauer. Constrained prfs for unbounded inputs with short keys. Cryptology ePrint Archive, Report 2016/279, 2016. http://eprint.iacr.org/.

[AFP14] Hamza Abusalah, Georg Fuchsbauer, and Krzysztof Pietrzak. Constrained prfs for unbounded inputs. Cryptology ePrint Archive, Report 2014/840, 2014. http://eprint.iacr.org/.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *Advances in Cryptology–CRYPTO 2015*, pages 308–326. Springer, 2015.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. Cryptology ePrint Archive, Report 2015/730, 2015. http://eprint.iacr.org/.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012.

[BFP+15] Abhishek Banerjee, Georg Fuchsbauer, Chris Peikert, Krzysztof Pietrzak, and Sophie Stevens. Key-homomorphic constrained pseudorandom functions. In *Theory of Cryptography*, pages 31–60. Springer, 2015.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Public-Key Cryptography–PKC 2014*, pages 501–519. Springer, 2014.

[BGJS15] Saikrishna Badrinarayanan, Divya Gupta, Abhishek Jain, and Amit Sahai. Multi-input functional encryption for unbounded arity functions. In *Advances in Cryptology–ASIACRYPT 2015*, pages 27–51. Springer, 2015.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions. In *Theory of Cryptography*, pages 1–30. Springer, 2015.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology-ASIACRYPT 2013*, pages 280–300. Springer, 2013.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *Advances in Cryptology–CRYPTO 2014*, pages 480–499. Springer, 2014.

[CRV14] Nishanth Chandran, Srinivasan Raghuraman, and Dhinakaran Vinayagamurthy. Constrained pseudorandom functions: Verifiable and delegatable. 2014. http://eprint.iacr.org/.

[DKW16] Apoorvaa Deshpande, Venkata Koppula, and Brent Waters. Constrained pseudorandom functions for unconstrained inputs. Cryptology ePrint Archive, Report 2016/301, 2016. http://eprint.iacr.org/.

[FKPR14] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained prfs. In *Advances in Cryptology–ASIACRYPT 2014*, pages 82–101. Springer, 2014.

[Fuc14] Georg Fuchsbauer. Constrained verifiable random functions. In *Security and Cryptography for Networks*, pages 95–114. Springer, 2014.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM on Symposium on Theory of Computing*, pages 169–178. ACM, 2009.

[GGH+13] Shelly Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Anant Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 40–49. IEEE, 2013.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[GJKS15] Vipul Goyal, Abhishek Jain, Venkata Koppula, and Amit Sahai. Functional encryption for randomized functionalities. In *Theory of Cryptography*, pages 325–351. Springer, 2015.

[GLSW15] Craig Gentry, Allison Bishop Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 151–170. IEEE, 2015.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, pages 99–108. ACM, 2011.

[HKKW14] Dennis Hofheinz, Akshay Kamath, Venkata Koppula, and Brent Waters. Adaptively secure constrained pseudorandom functions. 2014. `http://eprint.iacr.org/`.

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Adaptively secure puncturable pseudorandom functions in the standard model. In *Advances in Cryptology–ASIACRYPT 2015*, pages 79–102. Springer, 2015.

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In *Advances in Cryptology–EUROCRYPT 2014*, pages 201–220. Springer, 2014.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 163–172. ACM, 2015.

[IPS15] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In *Theory of Cryptography*, pages 668–697. Springer, 2015.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing*, pages 419–428. ACM, 2015.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 669–684. ACM, 2013.

[MPR11] Hemanta K Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-based signatures. In *Topics in Cryptology–CT-RSA 2011*, pages 376–392. Springer, 2011.

[MRV99] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 120–130. IEEE, 1999.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In *Advances in Cryptology–ASIACRYPT 2015*, pages 121–145. Springer, 2015.

[OT14] Tatsuaki Okamoto and Katsuyuki Takashima. Efficient attribute-based signatures for non-monotone predicates in the standard model. *Cloud Computing, IEEE Transactions on*, 2(4):409–421, 2014.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *Advances in Cryptology–CRYPTO 2014*, pages 500–517. Springer, 2014.

[SAH16] Yusuke Sakai, Nuttapong Attrapadung, and Goichiro Hanaoka. Attribute-based signatures for circuits from bilinear map. In *Public-Key Cryptography–PKC 2016*, pages 283–300. Springer, 2016.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 475–484. ACM, 2014.

[TLL14] Fei Tang, Hongda Li, and Bei Liang. Attribute-based signatures for circuits from multilinear maps. In *Information Security*, pages 54–71. Springer, 2014.

[Wat15] Brent Waters. A punctured programming approach to adaptively secure functional encryption. In *Advances in Cryptology–CRYPTO 2015*, pages 678–697. Springer, 2015.

Appendix A: Proof of Theorem 3.1

Theorem 3.1 (Security of the CPRF Construction of Section 3.4). *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a secure positional accumulator according to Definition 2.4,* ITR *is a secure cryptographic iterator as per Definition 2.5,* SPS *is a secure splittable signature scheme as defined in Definition 2.6, and* PRG *is a secure injective pseudorandom generator,the* CPRF *construction of Section 3.4 satisfies correctness under constraining and selective pseudorandomness as defined in Definition 3.1.*

Proof. Note that the *correctness under constraining* property of the CPRF construction of Section 3.4 follows immediately from its construction and the correctness of all the underlying cryptographic building blocks. In order to prove *selective pseudorandomness*, we will first introduce a sequence of hybrid experiments and next show based on the security properties of various primitives that the advantage of any PPT adversary $\mathcal{A}$ in any two neighboring hybrid experiments is negligibly different. Finally, we will argue that the advantage of any PPT adversary $\mathcal{A}$ in the final hybrid experiment is negligible. Observe that since we are working in the selective model, the challenger $\mathcal{B}$ knows the challenge input $x^* = x_0^* \dots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ and the SSB hash value $h^* = \mathcal{H}_{\text{HK}}(x^*)$ prior to receiving any constrained key query from $\mathcal{A}$. Lets assume that the total number of constrained key query made by $\mathcal{A}$ is $\hat{q}$. Note that we do not consider any evaluation query in view of Remark 3.1. The hybrid experiments are described below:

Sequence of Hybrid Experiments

Hyb_0: This experiment corresponds to the real selective pseudorandomness experiment described in Definition 3.1 of Section 3.1. More precisely, this experiment proceeds as follows:

- $\mathcal{A}$ submits the challenge input $x^* = x_0^* \dots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ to $\mathcal{B}$.
- $\mathcal{B}$ generates a master CPRF key $\text{SK}_{\text{CPRF}} = (K, \text{HK}) \xleftarrow{\$} \mathsf{CPRF.Setup}(1^\lambda)$ as described in Section 3.4. Next it selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, it computes $y^* = \mathsf{CPRF}(\text{SK}_{\text{CPRF}}, x^*) = \mathcal{F}(K, (h^* = \mathcal{H}_{\text{HK}}(x^*), \ell^*))$. On the other hand, if $b = 1$, it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{CPRF}}$. $\mathcal{B}$ provides y^* to $\mathcal{A}$.
- For, $\eta = 1, \dots, \hat{q}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} = \langle Q^{(\eta)}, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta^{(\eta)}, q_0^{(\eta)}, q_{\text{ACC}}^{(\eta)}, q_{\text{REJ}}^{(\eta)} \rangle \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ creates

$$\text{SK}_{\text{CPRF}}\{M^{(\eta)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K, K_1^{(\eta)}, \dots, K_\lambda^{(\eta)}, K_{\text{SPS},A}^{(\eta)}]) \end{array} \right)$$
$$\xleftarrow{\$} \mathsf{CPRF.Constrain}(\text{SK}_{\text{CPRF}}, M^{(\eta)}),$$

  as described in Section 3.4, and provides $\text{SK}_{\text{CPRF}}\{M^{(\eta)}\}$ to $\mathcal{A}$. Here we assign the index η to all those components of $\text{SK}_{\text{CPRF}}\{M^{(\eta)}\}$ which are generated during the execution of $\mathsf{CPRF.Constrain}(\text{SK}_{\text{CPRF}}, M^{(\eta)})$.
- $\mathcal{A}$ eventually outputs a guess bit $b' \in \{0,1\}$.

$\mathsf{Hyb}_{0,\nu}$ ($\nu = 1, \ldots, \hat{q}$): This experiment is identical to Hyb_0 except that for $\eta \in [\hat{q}]$, in reply to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ returns the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\eta)}\} = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]}) \end{pmatrix}$$

to $\mathcal{A}$, if $\underline{\eta \leq \nu}$, while the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\eta)}\} = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}]) \end{pmatrix}$$

to $\mathcal{A}$, if $\underline{\eta > \nu}$, where the program $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CPRF}}$ is a modification of the program $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}$ (Fig. 3.4) and is depicted in Fig. A.1. Observe that $\mathsf{Hyb}_{0,0}$ coincides with Hyb_0.

Hyb_1: This experiment corresponds to $\mathsf{Hyb}_{0,\hat{q}}$, i.e., in this experiment, for all $\eta = 1, \ldots, \hat{q}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ returns the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\eta)}\} = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]) \end{pmatrix}$$

to $\mathcal{A}$, while the rest of the experiment proceeds as in Hyb_0.

Hyb_2: This experiment proceeds as follows:

- After receiving the challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\mathrm{CPRF}}$ from $\mathcal{A}$, $\mathcal{B}$ performs the following:
  1. It first chooses a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$ and generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ as in Hyb_1.
  2. Next it computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$ and creates the punctured PPRF key $\underline{K\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K, (h^*, \ell^*))}$.
  3. Then, it selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, it computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. Otherwise, it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$.

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   (b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \ \wedge \ [(h, \ell_{\text{INP}}) \neq (h^*, \ell^*)]$, output $\mathcal{F}(K, (h, \ell_{\text{INP}}))$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, output $\perp$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.1. $\mathsf{Constrained\text{-}Key.Prog}'_{\text{CPRF}}$

4. It provides y^* to $\mathcal{A}$.

- For all $\eta = 1, \ldots, \hat{q}$, to answer the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with$M^{(\eta)}(x*) = 0$, $\mathcal{B}$ generates all the components as in Hyb_1, however, it sends back the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\eta)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\text{CPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K\{(h^*, \ell^*)\}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, \\ \qquad K_{\text{SPS},A}^{(\eta)}, h^*, \ell^*]) \end{array} \right)$$

to $\mathcal{A}$.

- At the end $\mathcal{A}$ outputs a guess bit $b' \in \{0, 1\}$ as usual.

Analysis

Let $\mathsf{Adv}_{\mathcal{A}}^{(0)}(\lambda), \mathsf{Adv}_{\mathcal{A}}^{(0,\nu)}(\lambda)(\nu = 1, \ldots, \hat{q})$, $\mathsf{Adv}_{\mathcal{A}}^{(1)}(\lambda)$, and $\mathsf{Adv}_{\mathcal{A}}^{(2)}(\lambda)$ represent respectively the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in $\mathsf{Hyb}_0, \mathsf{Hyb}_{0,\nu}(\nu = 1, \ldots, \hat{q})$, Hyb_1, and Hyb_2. From the description of the hybrid experiments it follows that $\mathsf{Adv}_{\mathcal{A}}^{\text{CPRF,SEL-PR}}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,0)}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(1)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\hat{q})}(\lambda)$. Therefore, we have

$$\mathsf{Adv}_{\mathcal{A}}^{\text{CPRF,SEL-PR}}(\lambda) \leq \sum_{\nu=1}^{\hat{q}} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu)}(\lambda)| + |\mathsf{Adv}_{\mathcal{A}}^{(1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(2)}(\lambda)| + \mathsf{Adv}_{\mathcal{A}}^{(2)}(\lambda). \quad \text{(A.1)}$$

Lemmas A.1–A.3 will show that the RHS of Eq. (A.1) is negligible and hence Theorem 3.1 follows. □

A.1 Lemmas for the Proof of Theorem 3.1

Lemma A.1. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$ satisfies the correctness under puncturing property defined in Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The only difference between Hyb_1 and Hyb_2 is the following: For $\eta = 1, \ldots, \hat{q}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, in Hyb_1, $\mathcal{B}$ includes $\mathcal{IO}(P_0^{(\eta)})$ within the constrained key $\text{SK}_{\text{CPRF}}\{M^{(\eta)}\}$ returned to $\mathcal{A}$, while in Hyb_2, $\mathcal{B}$ includes $\mathcal{IO}(P_1^{(\eta)})$ instead, where

- $P_0^{(\eta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\text{CPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}^{(\eta)}_{\text{ACC}}, \text{PP}^{(\eta)}_{\text{ITR}}, K, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K^{(\eta)}_{\text{SPS},A}, h^*, \ell^*]$,
- $P_1^{(\eta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\text{CPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}^{(\eta)}_{\text{ACC}}, \text{PP}^{(\eta)}_{\text{ITR}}, K\{(h^*, \ell^*)\}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K^{(\eta)}_{\text{SPS},A}, h^*, \ell^*]$.

Here, $K\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K, (h^*, \ell^*))$ and program $\mathsf{Constrained\text{-}Key.Prog}'_{\text{CPRF}}$ is depicted in Fig. A.1.

Now observe that the program $\mathsf{Constrained\text{-}Key.Prog}'_{\text{CPRF}}$ computes $\mathcal{F}(K, (h, \ell_{\text{INP}}))$ if and only if $(h, \ell_{\text{INP}}) \neq (h^*, \ell^*)$. As a consequence, by the correctness under puncturing property of the PPRF $\mathcal{F}$, the functionality of the program $\mathsf{Constrained\text{-}Key.Prog}'_{\text{CPRF}}$ does not change if the punctured PPRF key $K\{(h^*, \ell^*)\}$ is hardwired into it in place of the full PPRF key K.

Therefore, by the security of $\mathcal{IO}$, Lemma A.1 follows. Ofcourse, for achieving the result we would have to consider a sequence of $\hat{q}$ hybrid experiments where in each hybrid experiment we change the hardwiring of the program $\mathsf{Constrained\text{-}Key.Prog}'_{\text{CPRF}}$ included in the η^{th} queried constrained key, for $\eta = 1, \ldots, \hat{q}$. □

Lemma A.2. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $\mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $\mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda)$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x^*_0 \ldots x^*_{\ell^*-1} \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$. $\mathcal{B}$ will *implicitly* view the key K^* as the key K.
  2. $\mathcal{B}$ sends (h^*, ℓ^*) as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back a punctured PPRF key $K^*\{(h^*, \ell^*)\}$ and a value $r^* \in \mathcal{Y}_{\text{PPRF}}$, where either $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$ or $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
  3. $\mathcal{B}$ returns the challenge CPRF value $y^* = r^*$ to $\mathcal{A}$.
- For $\eta = 1, \ldots, \hat{q}$, in response to the η^{th} constrained query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K^{(\eta)}_{\text{SPS},A}, K^{(\eta)}_{\text{SPS},E} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it creates $(\text{PP}^{(\eta)}_{\text{ACC}}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}^{(\eta)}_{\text{ITR}}, v_0^{(\eta)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.

3. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\eta)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K^*\{(h^*, \ell^*)\}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, \\ \quad K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its PPRF selective pseudorandomness experiment.

Note that the simulation of Hyb_2 by $\mathcal{B}$ is perfect. Also, if $\mathcal{A}$ wins in this simulated Hyb_2, then $\mathcal{B}$ wins in the PPRF selective pseudorandomness experiment against $\mathcal{F}$. This completes the proof of Lemma A.2. □

Lemma A.3. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a secure positional accumulator as defined in Definition 2.4,* ITR *is a secure cryptographic iterator according to Definition 2.5,* SPS *is a secure splitable signature as per Definition 2.6, and* PRG *is a secure injective pseudorandom generator, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of our Lemma A.3 extends the ideas involved in the security proof for the message-hiding encoding of [KLW15]. Lemma 1 in the security proof of the CPRF construction of [DKW16] also employs a similar technique. However, as mentioned earlier, making the use of the somewhere statistically binding hash function and suitably modifying the construction of [DKW16] we are able to remove the flaw in the argument of [DKW16]. We will first provide a complete description of the sequence of hybrid experiments involved in the proof of our Lemma A.3 and then provide the analysis of those hybrid experiments providing the details for only those segments which are distinct from [DKW16].

Let $t^{*(\nu)}$ denotes the running time of the TM $M^{(\nu)} \in \mathbb{M}_\lambda$, queried by the adversary $\mathcal{A}$, on input the challenge string x^* and $2^{\tau^{*(\nu)}}$ be the smallest power of two greater than $t^{*(\nu)}$. The sequence of intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1}$ and $\mathsf{Hyb}_{0,\nu}$ are described below:

Sequence of Intermediate Hybrids between $\mathsf{Hyb}_{0,\nu-1}$ and $\mathsf{Hyb}_{0,\nu}$

$\mathsf{Hyb}_{0,\nu-1,0}$: This experiment coincides with $\mathsf{Hyb}_{0,\nu-1}$.

$\mathsf{Hyb}_{0,\nu-1,1}$: This experiment if analogous to $\mathsf{Hyb}_{0,\nu-1,0}$ except that to answer the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first picks PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next, it generates $(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$.

3. It provides $\mathcal{A}$ with the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, \\ \quad \underline{K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*}]) \end{array} \right),$$

where the program $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}$, which is formed by modifying the program $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CPRF}}$ (Fig. A.1), is depicted in Fig. A.2.

Constants: TM $M = \langle Q, \Sigma_{\mathrm{INP}}, \Gamma_{\mathrm{TAPE}}, \delta, q_0, q_{\mathrm{AC}}, q_{\mathrm{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator $\mathrm{PP}_{\mathrm{ACC}}$, Public parameters for iterator $\mathrm{PP}_{\mathrm{ITR}}$, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\mathrm{SPS},A}$, $K_{\mathrm{SPS},B}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String $\mathrm{SEED}_{\mathrm{IN}}$, Header position $\mathrm{POS}_{\mathrm{IN}}$, Symbol $\mathrm{SYM}_{\mathrm{IN}}$, TM state $\mathrm{ST}_{\mathrm{IN}}$, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\mathrm{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\mathrm{INP}}))$, or Header Position ($\mathrm{POS}_{\mathrm{OUT}}$, Symbol $\mathrm{SYM}_{\mathrm{OUT}}$, TM state $\mathrm{ST}_{\mathrm{OUT}}$, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\mathrm{SPS,OUT}}$, String $\mathrm{SEED}_{\mathrm{OUT}}$), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$. If $[\mathsf{PRG}(\mathrm{SEED}_{\mathrm{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\mathrm{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\mathrm{PP}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}}, \pi_{\mathrm{ACC}}) = 0$, output $\perp$.

3.(a) Compute $r_{\mathrm{SPS},A} = \mathcal{F}(K_{\mathrm{SPS},A}, (h, \ell_{\mathrm{INP}}, t-1))$, $(\mathrm{SK}_{\mathrm{SPS},A}, \mathrm{VK}_{\mathrm{SPS},A}, \mathrm{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\mathrm{SPS},A})$.

(b) Compute $r_{\mathrm{SPS},B} = \mathcal{F}(K_{\mathrm{SPS},B}, (h, \ell_{\mathrm{INP}}, t-1))$, $(\mathrm{SK}_{\mathrm{SPS},B}, \mathrm{VK}_{\mathrm{SPS},B}, \mathrm{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\mathrm{SPS},B})$.

(c) Set $m_{\mathrm{IN}} = (v_{\mathrm{IN}}, \mathrm{ST}_{\mathrm{IN}}, w_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}})$ and $\alpha =$'-'.

(d) If $\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},A}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1$, set $\alpha =$'A'.

(e) If $[\alpha =$'-'$] \ \wedge \ [(t > t^*) \ \vee \ (h \neq h^*) \ \vee \ (\ell_{\mathrm{INP}} \neq \ell^*)]$, output $\perp$.
Else if $[\alpha =$'-'$] \ \wedge \ [\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},B}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1]$, set $\alpha =$'B'.

(f) If $\alpha =$'-', output $\perp$.

4.(a) Compute $(\mathrm{ST}_{\mathrm{OUT}}, \mathrm{SYM}_{\mathrm{OUT}}, \beta) = \delta(\mathrm{ST}_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}})$ and $\mathrm{POS}_{\mathrm{OUT}} = \mathrm{POS}_{\mathrm{IN}} + \beta$.

(b) If $\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{REJ}}$, output $\perp$.
Else if $[\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{AC}}] \ \wedge \ [\alpha =$'$B$'$]$, output $\perp$.
Else if $\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{AC}}$, output $\mathcal{F}(K, (h, \ell_{\mathrm{INP}}))$.

5.(a) Compute $w_{\mathrm{OUT}} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{OUT}}, \mathrm{POS}_{\mathrm{IN}}, \mathrm{AUX})$. If $w_{\mathrm{OUT}} = \perp$, output $\perp$.

(b) Compute $v_{\mathrm{OUT}} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}, v_{\mathrm{IN}}, (\mathrm{ST}_{\mathrm{IN}}, w_{\mathrm{IN}}, \mathrm{POS}_{\mathrm{IN}}))$.

6.(a) Compute $r'_{\mathrm{SPS},A} = \mathcal{F}(K_{\mathrm{SPS},A}, (h, \ell_{\mathrm{INP}}, t))$, $(\mathrm{SK}'_{\mathrm{SPS},A}, \mathrm{VK}'_{\mathrm{SPS},A}, \mathrm{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\mathrm{SPS},A})$.

(b) Compute $r'_{\mathrm{SPS},B} = \mathcal{F}(K_{\mathrm{SPS},B}, (h, \ell_{\mathrm{INP}}, t))$, $(\mathrm{SK}'_{\mathrm{SPS},B}, \mathrm{VK}'_{\mathrm{SPS},B}, \mathrm{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\mathrm{SPS},B})$.

(c) Set $m_{\mathrm{OUT}} = (v_{\mathrm{OUT}}, \mathrm{ST}_{\mathrm{OUT}}, w_{\mathrm{OUT}}, \mathrm{POS}_{\mathrm{OUT}})$.
Compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},\alpha}, m_{\mathrm{OUT}})$.

7. If $t + 1 = 2^{\tau'}$, set $\mathrm{SEED}_{\mathrm{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\mathrm{INP}}))$.
Else, set $\mathrm{SEED}_{\mathrm{OUT}} = \epsilon$.
8. Output $(\mathrm{POS}_{\mathrm{OUT}}, \mathrm{SYM}_{\mathrm{OUT}}, \mathrm{ST}_{\mathrm{OUT}}, w_{\mathrm{OUT}}, v_{\mathrm{OUT}}, \sigma_{\mathrm{SPS,OUT}}, \mathrm{SEED}_{\mathrm{OUT}})$.

Fig. A.2. $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}$

$\mathsf{Hyb}_{\mathbf{0},\nu-\mathbf{1},\mathbf{2}}$: In this experiment, in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ performs the following steps:

1. It first chooses PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}$, $K_{\mathrm{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.

2. Next, it generates $(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$.
3. It provides $\mathcal{A}$ with the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\overline{\mathsf{Accumulate.Prog}^{(1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]}), \\ \mathcal{IO}(\overline{\mathsf{Change\text{-}SPS.Prog}^{(1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]}), \\ \mathcal{IO}(\overline{\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]}) \end{array} \right),$$

where the programs $\mathsf{Accumulate.Prog}^{(1)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(1)}$ are modifications of the programs $\mathsf{Accumulate.Prog}$ and $\mathsf{Change\text{-}SPS.Prog}$ (Figs. 3.2 and 3.3) and are depicted in Figs. A.3 and A.4 respectively.

The rest of the experiment proceeds in the same way as in $\mathsf{Hyb}_{0,\nu-1,1}$.

$\mathsf{Hyb}_{0,\nu-1,3}$: In this experiment, to answer the ν^{th} constrained key query of $\mathcal{A}$ correspond-

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator $\mathrm{PP}_{\mathrm{ACC}}$, Public parameters for iterator $\mathrm{PP}_{\mathrm{ITR}}$, PPRF keys $K_{\mathrm{SPS},E}$, $K_{\mathrm{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol $\mathrm{SYM}_{\mathrm{IN}}$, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) Compute $r_{\mathrm{SPS},E} = \mathcal{F}(K_{\mathrm{SPS},E}, (h,i))$, $(\mathrm{SK}_{\mathrm{SPS},E}, \mathrm{VK}_{\mathrm{SPS},E}, \mathrm{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\mathrm{SPS},E})$.
(b) Compute $r_{\mathrm{SPS},F} = \mathcal{F}(K_{\mathrm{SPS},F}, (h,i))$, $(\mathrm{SK}_{\mathrm{SPS},F}, \mathrm{VK}_{\mathrm{SPS},F}, \mathrm{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\mathrm{SPS},F})$.
(c) Set $m_{\mathrm{IN}} = (v_{\mathrm{IN}}, \mathrm{ST}, w_{\mathrm{IN}}, 0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},E}, m_{\mathrm{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =$'-'$] \wedge [(i \neq \ell^*) \vee (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},F}, m_{\mathrm{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\mathrm{HK}, h, i, \mathrm{SYM}_{\mathrm{IN}}, \pi_{\mathrm{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\mathrm{OUT}} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}}, i, \mathrm{AUX})$. If $w_{\mathrm{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\mathrm{OUT}} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}, v_{\mathrm{IN}}, (\mathrm{ST}, w_{\mathrm{IN}}, 0))$.
4.(a) Compute $r'_{\mathrm{SPS},E} = \mathcal{F}(K_{\mathrm{SPS},E}, (h, i+1))$, $(\mathrm{SK}'_{\mathrm{SPS},E}, \mathrm{VK}'_{\mathrm{SPS},E}, \mathrm{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\mathrm{SPS},E})$.
(b) Set $m_{\mathrm{OUT}} = (v_{\mathrm{OUT}}, \mathrm{ST}, w_{\mathrm{OUT}}, 0)$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},E}, m_{\mathrm{OUT}})$.
5. Output $(w_{\mathrm{OUT}}, v_{\mathrm{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.3. $\mathsf{Accumulate.Prog}^{(1)}$

ing to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the PPRF keys as well as the public parameters for the positional accumulator and iterator just as in $\mathsf{Hyb}_{0,\nu-1,2}$, however, it returns

Constants: PPRF keys $K_{\text{SPS},A}$, $K_{\text{SPS},B}$, $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m = (v, \text{ST}, w, 0)$ and α='-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set α ='E'.
(e) If $[\alpha$ ='-'$] \wedge [(\ell_{\text{INP}} \neq \ell^*) \vee (h \neq h^*)]$, output $\perp$.
Else if $[\alpha$ ='-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m, \sigma_{\text{SPS,IN}}) = 1]$, set α ='F'.
(f) If α ='-', output $\perp$.
2.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
(b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{DSPS},B})$.
(c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [\alpha$ ='F'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.4. $\mathsf{Change\text{-}SPS.Prog}^{(1)}$

the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, \\ \quad K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right),$$

where the programs $\mathsf{Accumulate.Prog}^{(2)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(2)}$ are modifications of the programs $\mathsf{Accumulate.Prog}^{(1)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(1)}$ (Figs. A.3 and A.4) and are depicted in Figs. A.5 and A.6 respectively. The remaining part of the experiment is similar to $\mathsf{Hyb}_{0,\nu-1,2}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota}$ ($\iota = 0, \ldots, \ell^* - 1$): In this hybrid experiment, to answer the ν^{th}constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,3}$.
2. Next, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $j = 1, \ldots, \iota$, it iteratively computes the following:
   - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1)$
   - $w_j^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
   - $\text{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
   - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$

   It sets $m_{\iota,0}^{(\nu)} = (v_\iota^{(\nu)}, q_0^{(\nu)}, w_\iota^{(\nu)}, 0)$.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =\text{'-'}] \ \wedge\ [\underline{(i > \ell^*)\ \vee\ (i = 0)}\ \vee\ (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =\text{'-'}] \ \wedge\ [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4.(a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
(b) $\underline{\text{Compute } r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i+1)), (\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},F}).}$
(c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$.
$\underline{\text{If } i < \ell^*, \text{ compute } \sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}}).}$
$\underline{\text{Else, compute } \sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}}).}$
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.5. $\mathsf{Accumulate.Prog}^{(2)}$

Constants: PPRF keys $K_{\text{SPS},A}$, $K_{\text{SPS},B}$, $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m = (v, \text{ST}, w, 0)$ and α='-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =\text{'-'}] \ \wedge\ [\underline{(\ell_{\text{INP}} > \ell^*)}\ \vee\ (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =\text{'-'}] \ \wedge\ [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
(b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{DSPS},B})$.
(c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \ \wedge\ [\alpha =$'$F$'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.6. $\mathsf{Change\text{-}SPS.Prog}^{(2)}$

3. It gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, \\ \quad K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right),$$

where the programs $\mathsf{Accumulate.Prog}^{(3,\iota)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}$ are alterations of the programs $\mathsf{Accumulate.Prog}^{(2)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(2)}$ (Figs. A.5 and A.6) and are described in Figs. A.7 and A.8 respectively.

The remaining part of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,3}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota'}$ $(\iota = 0, \ldots, \ell^* - 1)$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,3}$ except that in re-

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K_{\text{SPS},E}, K_{\text{SPS},F}$, Message $m_{\iota,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h,i)), (\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h,i)), (\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =$'-'$] \wedge [(i > \ell^*) \vee \underline{(0 \le i \le \iota)} \vee (h \ne h^*)]$, output $\perp$.
Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4.(a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1)), (\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
(b) Compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i+1)), (\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},F})$.
(c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$.
If $[(h,i) = (h^*, \iota)] \wedge [m_{\text{IN}} = m_{\iota,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
Else if $[(h,i) = (h^*, \iota)] \wedge [m_{\text{IN}} \ne m_{\iota,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.7. $\mathsf{Accumulate.Prog}^{(3,\iota)}$

sponse to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,3}$.
2. Next, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$. For $\underline{j = 1, \ldots, \iota+1}$, it iteratively computes the following:
   - $\text{AUX}^{(\nu)}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1)$
   - $w^{(\nu)}_j = \mathsf{ACC.Update}(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_{j-1}, x^*_{j-1}, j-1, \text{AUX}^{(\nu)}_j)$
   - $\text{STORE}^{(\nu)}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1, x^*_{j-1})$
   - $v^{(\nu)}_j = \mathsf{ITR.Iterate}(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_{j-1}, (q^{(\nu)}_0, w^{(\nu)}_{j-1}, 0))$

   It sets $\underline{m^{(\nu)}_{\iota+1,0} = (v^{(\nu)}_{\iota+1}, q^{(\nu)}_0, w^{(\nu)}_{\iota+1}, 0)}$.

Constants: PPRF keys $K_{\text{SPS},A}$, $K_{\text{SPS},B}$, $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m = (v, \text{ST}, w, 0)$ and α='-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set α ='E'.
(e) If $[\alpha$ ='-'$]\ \wedge\ [(\ell_{\text{INP}} > \ell^*)\ \vee\ (0 < \ell_{\text{INP}} \leq \iota)\ \vee\ (h \neq h^*)]$, output $\perp$.
Else if $[\alpha$ ='-'$]\ \wedge\ [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m, \sigma_{\text{SPS,IN}}) = 1]$, set α ='F'.
(f) If α ='-', output $\perp$.
2.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
(b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{DSPS},B})$.
(c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)]\ \wedge\ [\alpha$ ='F'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.8. $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}$

3. It gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota')}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, \\ \quad K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right),$$

where the program $\mathsf{Accumulate.Prog}^{(3,\iota')}$ is an alteration of the program $\mathsf{Accumulate.Prog}^{(3,\iota)}$ (Fig. A.7) and is shown in Fig. A.9.

$\mathsf{Hyb}_{0,\nu-1,4}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,3,(\ell^*-1)'}$ with the exception that now in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ does not generate the PPRF key $K_{\text{SPS},F}^{(\nu)}$ and gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, \\ \quad K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right),$$

where the program Accumulate.Prog is shown in Fig. 3.2 while the program $\mathsf{Change\text{-}SPS.Prog}^{(4)}$, which is a modification of the program $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}$ (Fig. A.8), is depicted in Fig. A.10.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K_{\text{SPS},E}, K_{\text{SPS},F}$, Message $m_{\iota+1,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h,i)), (\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h,i)), (\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =\text{'-'}] \ \wedge\ [(i > \ell^*) \ \vee\ (0 \le i \le \iota) \ \vee\ (h \ne h^*)]$, output $\perp$.
Else if $[\alpha =\text{'-'}] \ \wedge\ [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4.(a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1)), (\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
(b) Compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i+1)), (\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},F})$.
(c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$.
If $[(h,i) = (h^*, \iota)] \ \wedge\ [m_{\text{OUT}} = m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
Else if $[(h,i) = (h^*, \iota)] \ \wedge\ [m_{\text{OUT}} \ne m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E} m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.9. $\mathsf{Accumulate.Prog}^{(3,\iota')}$

Constants: PPRF keys $K_{\text{SPS},A}, K_{\text{SPS},B}, K_{\text{SPS},E}$, Message $m_{\ell^*,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}), (\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Set $m = (v, \text{ST}, w, 0)$.
(c) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
2.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0)), (\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
(b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0)), (\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},B})$.
(c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \ \wedge\ [m \ne m_{\ell^*,0}]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.10. $\mathsf{Change\text{-}SPS.Prog}^{(4)}$

$\mathsf{Hyb}_{0,\nu-1,4,\gamma}$ ($\gamma = 1, \ldots, t^{*(\nu)} - 1$): This experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4}$ except that in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,4}$.
2. Next, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$. For $j = 1, \ldots, \ell^*$, it iteratively computes the following:
   - $\text{AUX}^{(\nu)}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1)$
   - $w^{(\nu)}_j = \mathsf{ACC.Update}(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_{j-1}, x^*_{j-1}, j-1, \text{AUX}^{(\nu)}_j)$
   - $\text{STORE}^{(\nu)}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1, x^*_{j-1})$
   - $v^{(\nu)}_j = \mathsf{ITR.Iterate}(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_{j-1}, (q^{(\nu)}_0, w^{(\nu)}_{j-1}, 0))$

3. Then, $\mathcal{B}$ sets $\mathrm{ST}_{M^{(\nu)},0} = q_0^{(\nu)}, \mathrm{POS}_{M^{(\nu)},0} = 0$, and for $t = 1, \ldots, \gamma$, computes the following:
   - $(\mathrm{SYM}_{M^{(\nu)},t-1}, \pi^{(\nu)}_{\mathrm{ACC},t-1}) = \mathsf{ACC.Prep\text{-}Read}(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{STORE}^{(\nu)}_{\ell^*+t-1}, \mathrm{POS}_{M^{(\nu)},t-1})$
   - $\mathrm{AUX}^{(\nu)}_{\ell^*+t} = \mathsf{ACC.Prep\text{-}Write}(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{STORE}^{(\nu)}_{\ell^*+t-1}, \mathrm{POS}_{M^{(\nu)},t-1})$
   - $(\mathrm{ST}_{M^{(\nu)},t}, \mathrm{SYM}^{(\mathrm{WRITE})}_{M^{(\nu)},t}, \beta) = \delta^{(\nu)}(\mathrm{ST}_{M^{(\nu)},t-1}, \mathrm{SYM}_{M^{(\nu)},t-1})$
   - $w^{(\nu)}_{\ell^*+t} = \mathsf{ACC.Update}(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, w^{(\nu)}_{\ell^*+t-1}, \mathrm{SYM}^{(\mathrm{WRITE})}_{M^{(\nu)},t}, \mathrm{POS}_{M^{(\nu)},t-1}, \mathrm{AUX}^{(\nu)}_{\ell^*+t})$
   - $v^{(\nu)}_{\ell^*+t} = \mathsf{ITR.Iterate}(\mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, v^{(\nu)}_{\ell^*+t-1}, (\mathrm{ST}_{M^{(\nu)},t-1}, w^{(\nu)}_{\ell^*+t-1}, \mathrm{POS}_{M^{(\nu)},t-1}))$
   - $\mathrm{STORE}^{(\nu)}_{\ell^*+t} = \mathsf{ACC.Write\text{-}Store}(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{STORE}^{(\nu)}_{\ell^*+t-1}, \mathrm{POS}_{M^{(\nu)},t-1}, \mathrm{SYM}^{(\mathrm{WRITE})}_{M^{(\nu)},t})$
   - $\mathrm{POS}_{M^{(\nu)},t} = \mathrm{POS}_{M^{(\nu)},t-1} + \beta$

   $\mathcal{B}$ sets $m^{(\nu)}_{\ell^*,\gamma} = (v^{(\nu)}_{\ell^*+\gamma}, \mathrm{ST}_{M^{(\nu)},\gamma}, w^{(\nu)}_{\ell^*+\gamma}, \mathrm{POS}_{M^{(\nu)},\gamma})$.
4. $\mathcal{B}$ provides $\mathcal{A}$ with the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, w^{(\nu)}_0, \mathrm{STORE}^{(\nu)}_0, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\mathrm{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K^{(\nu)}_{\mathrm{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\nu)}_{\mathrm{SPS},A}, K^{(\nu)}_{\mathrm{SPS},E}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma)}_{\mathrm{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\mathrm{SPS},A}, \\ \quad K^{(\nu)}_{\mathrm{SPS},B}, m^{(\nu)}_{\ell^*,\gamma}, h^*, \ell^*])) \end{array}\right),$$

   where program $\mathsf{Change\text{-}SPS.Prog}$ is described in Fig. 3.3 and program $\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma)}_{\mathrm{CPRF}}$, a modification of program $\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\mathrm{CPRF}}$ (Fig. A.2), is described in Fig. A.11.

$\mathsf{Hyb}_{0,\nu-1,4,\gamma'}$ $(\gamma = 0, \ldots, t^{*(\nu)} - 1)$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,4}$ except that in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,4}$.
2. Next, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$. For $j = 1, \ldots, \ell^*$, it iteratively computes the following:
   - $\mathrm{AUX}^{(\nu)}_j = \mathsf{ACC.Prep\text{-}Write}(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{STORE}^{(\nu)}_{j-1}, j-1)$
   - $w^{(\nu)}_j = \mathsf{ACC.Update}(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, w^{(\nu)}_{j-1}, x^*_{j-1}, j-1, \mathrm{AUX}^{(\nu)}_j)$
   - $\mathrm{STORE}^{(\nu)}_j = \mathsf{ACC.Write\text{-}Store}(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{STORE}^{(\nu)}_{j-1}, j-1, x^*_{j-1})$
   - $v^{(\nu)}_j = \mathsf{ITR.Iterate}(\mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, v^{(\nu)}_{j-1}, (q^{(\nu)}_0, w^{(\nu)}_{j-1}, 0))$
3. Then, $\mathcal{B}$ sets $\mathrm{ST}_{M^{(\nu)},0} = q^{(\nu)}_0, \mathrm{POS}_{M^{(\nu)},0} = 0$, and for $t = 1, \ldots, \gamma$, computes the following:
   - $(\mathrm{SYM}_{M^{(\nu)},t-1}, \pi^{(\nu)}_{\mathrm{ACC},t-1}) = \mathsf{ACC.Prep\text{-}Read}(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{STORE}^{(\nu)}_{\ell^*+t-1}, \mathrm{POS}_{M^{(\nu)},t-1})$
   - $\mathrm{AUX}^{(\nu)}_{\ell^*+t} = \mathsf{ACC.Prep\text{-}Write}(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{STORE}^{(\nu)}_{\ell^*+t-1}, \mathrm{POS}_{M^{(\nu)},t-1})$
   - $(\mathrm{ST}_{M^{(\nu)},t}, \mathrm{SYM}^{(\mathrm{WRITE})}_{M^{(\nu)},t}, \beta) = \delta^{(\nu)}(\mathrm{ST}_{M^{(\nu)},t-1}, \mathrm{SYM}_{M^{(\nu)},t-1})$
   - $w^{(\nu)}_{\ell^*+t} = \mathsf{ACC.Update}(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, w^{(\nu)}_{\ell^*+t-1}, \mathrm{SYM}^{(\mathrm{WRITE})}_{M^{(\nu)},t}, \mathrm{POS}_{M^{(\nu)},t-1}, \mathrm{AUX}^{(\nu)}_{\ell^*+t})$
   - $v^{(\nu)}_{\ell^*+t} = \mathsf{ITR.Iterate}(\mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, v^{(\nu)}_{\ell^*+t-1}, (\mathrm{ST}_{M^{(\nu)},t-1}, w^{(\nu)}_{\ell^*+t-1}, \mathrm{POS}_{M^{(\nu)},t-1}))$
   - $\mathrm{STORE}^{(\nu)}_{\ell^*+t} = \mathsf{ACC.Write\text{-}Store}(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{STORE}^{(\nu)}_{\ell^*+t-1}, \mathrm{POS}_{M^{(\nu)},t-1}, \mathrm{SYM}^{(\mathrm{WRITE})}_{M^{(\nu)},t})$
   - $\mathrm{POS}_{M^{(\nu)},t} = \mathrm{POS}_{M^{(\nu)},t-1} + \beta$

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},B}$, Message $m_{\ell^*,\gamma}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \le t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},B})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \ \wedge \ [(t > t^*) \ \underline{\vee \ (t \le \gamma)} \ \vee \ (h \neq h^*) \ \vee \ (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \ \wedge \ [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \ \wedge \ [\alpha =$'$B$'$]$, output $\perp$.
   $\underline{\text{Else if } [\text{ST}_{\text{OUT}} = q_{\text{AC}}] \ \wedge \ [\alpha =\text{'}A\text{'}] \ \wedge \ [(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \ \wedge \ [t \le \gamma], \text{ output } \perp.}$
   $\underline{\text{Else if } \text{ST}_{\text{OUT}} = q_{\text{AC}}, \text{ output } \mathcal{F}(K, (h, \ell_{\text{INP}})).}$
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   $\underline{\text{If } [(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, \gamma)] \ \wedge \ [m_{\text{OUT}} = m_{\ell^*,\gamma}], \text{ compute } \sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}}).}$
   $\underline{\text{Else if } [(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, \gamma)] \ \wedge \ [m_{\text{OUT}} \neq m_{\ell^*,\gamma}], \text{ compute } \sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},B}, m_{\text{OUT}}).}$
   $\underline{\text{Else, compute } \sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}}).}$
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.11. $\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma)}_{\text{CPRF}}$

$\mathcal{B}$ sets $m^{(\nu)}_{\ell^*,\gamma} = (v^{(\nu)}_{\ell^*+\gamma}, \text{ST}_{M^{(\nu)},\gamma}, w^{(\nu)}_{\ell^*+\gamma}, \text{POS}_{M^{(\nu)},\gamma})$.

4. $\mathcal{B}$ provides $\mathcal{A}$ with the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma')}_{\text{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, \\ \quad \underline{K^{(\nu)}_{\text{SPS},B}, m^{(\nu)}_{\ell^*,\gamma}, h^*, \ell^*}]) \end{array} \right),$$

where program $\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma')}_{\text{CPRF}}$ is an alteration of program $\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma)}_{\text{CPRF}}$ (Fig. A.11) and is described in Fig. A.12.

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},B}$, Message $m_{\ell^*,\gamma}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},B})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \ \wedge \ [(t > t^*) \ \vee \ \underline{(t \leq \gamma + 1)} \ \vee \ (h \neq h^*) \ \vee \ (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \ \wedge \ [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \ \wedge \ [\alpha =$'$B$'$]$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \ \wedge \ [\alpha =$'$A$'$] \ \wedge \ [(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \ \wedge \ \underline{[t \leq \gamma + 1]}$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, output $\mathcal{F}(K, (h, \ell_{\text{INP}}))$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   $\underline{\text{If } [(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, \gamma + 1)] \ \wedge \ [m_{\text{IN}} = m_{\ell^*,\gamma}], \text{ compute } \sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}}).}$
   $\underline{\text{Else if } [(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, \gamma + 1)] \ \wedge \ [m_{\text{IN}} \neq m_{\ell^*,\gamma}], \text{ compute } \sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},B}, m_{\text{OUT}}).}$
   $\underline{\text{Else, compute } \sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}}).}$
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.12. $\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma')}_{\text{CPRF}}$

$\mathsf{Hyb}_{0,\nu-1,5}$: This experiment is similar to $\mathsf{Hyb}_{0,\nu-1,4,(t^{*(\nu)}-1)'}$ with the exception that in responding to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(3)}_{\text{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, \\ \quad \underline{K^{(\nu)}_{\text{SPS},B}, h^*, \ell^*}]) \end{array} \right),$$

where program $\mathsf{Constrained\text{-}Key.Prog}^{(3)}_{\text{CPRF}}$ is a modification of program $\mathsf{Constrained\text{-}Key.Prog}^{(2,\gamma')}_{\text{CPRF}}$ (Fig. A.12) and is described in Fig. A.13.

$\mathsf{Hyb}_{0,\nu-1,6}$: This experiment corresponds to $\mathsf{Hyb}_{0,\nu}$.

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},B}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$. If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
(b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
(b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \ \wedge \ [(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \ \wedge \ [t \leq t^*]$, output $\perp$.
Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, output $\mathcal{F}(K, (h, \ell_{\text{INP}}))$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
(b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},B})$.
(c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
If $(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, t^*)$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},B}, m_{\text{OUT}})$.
Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.13. $\mathsf{Constrained\text{-}Key.Prog}^{(3)}_{\text{CPRF}}$

Analysis

Let $\mathsf{Adv}^{(0,\nu-1,0)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(0,\nu-1,1)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(0,\nu-1,2)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(0,\nu-1,3)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(0,\nu-1,3,\iota)}_{\mathcal{A}}(\lambda)$ $(\iota = 0, \ldots, \ell^* - 1), \mathsf{Adv}^{(0,\nu-1,3,\iota')}_{\mathcal{A}}(\lambda)$ $(\iota = 0, \ldots, \ell^* - 1), \mathsf{Adv}^{(0,\nu-1,4)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(0,\nu-1,4,\gamma)}_{\mathcal{A}}(\lambda)$ $(\gamma = 1, \ldots, t^{*(\nu)} - 1)$, $\mathsf{Adv}^{(0,\nu-1,4,\gamma')}_{\mathcal{A}}(\lambda)$ $(\gamma = 0, \ldots, t^{*(\nu)} - 1), \mathsf{Adv}^{(0,\nu-1,5)}_{\mathcal{A}}(\lambda)$, and $\mathsf{Adv}^{(0,\nu-1,6)}_{\mathcal{A}}(\lambda)$ represent respectively the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in the hybrid experiment Hyb_Υ with Υ as indicated in the superscript of the advantage notation. By the description of the hybrid experiments it follows that $\mathsf{Adv}^{(0,\nu-1)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,\nu-1,0)}_{\mathcal{A}}(\lambda)$ and $\mathsf{Adv}^{(0,\nu)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,\nu-1,6)}_{\mathcal{A}}(\lambda)$. Thus we have,

$$
\begin{aligned}
&|\mathsf{Adv}^{(0,\nu-1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu)}_{\mathcal{A}}(\lambda)| \ \leq \\
&|\mathsf{Adv}^{(0,\nu-1,0)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,1)}_{\mathcal{A}}(\lambda)| + |\mathsf{Adv}^{(0,\nu-1,1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,2)}_{\mathcal{A}}(\lambda)| + \\
&|\mathsf{Adv}^{(0,\nu-1,2)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3)}_{\mathcal{A}}(\lambda)| + |\mathsf{Adv}^{(0,\nu-1,3)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,0)}_{\mathcal{A}}(\lambda)| + \\
&\sum_{\iota=0}^{\ell^*-1} |\mathsf{Adv}^{(0,\nu-1,3,\iota)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota')}_{\mathcal{A}}(\lambda)| + \sum_{\iota=0}^{\ell^*-2} |\mathsf{Adv}^{(0,\nu-1,3,\iota')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota+1)}_{\mathcal{A}}(\lambda)| + \\
&|\mathsf{Adv}^{(0,\nu-1,3,(\ell^*-1)')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4)}_{\mathcal{A}}(\lambda)| + |\mathsf{Adv}^{(0,\nu-1,4)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4,0')}_{\mathcal{A}}(\lambda)| + \\
&\sum_{\gamma=1}^{t^{*(\nu)}-1} |\mathsf{Adv}^{(0,\nu-1,4,(\gamma-1)')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4,\gamma)}_{\mathcal{A}}(\lambda)| + \sum_{\gamma=1}^{t^{*(\nu)}-1} |\mathsf{Adv}^{(0,\nu-1,4,\gamma)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4,\gamma')}_{\mathcal{A}}(\lambda)| + \\
&|\mathsf{Adv}^{(0,\nu-1,4,(t^{*(\nu)}-1)')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,5)}_{\mathcal{A}}(\lambda)| + |\mathsf{Adv}^{(0,\nu-1,5)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,6)}_{\mathcal{A}}(\lambda)|.
\end{aligned}
\tag{A.2}
$$

Lemmas A.4–A.15 will prove that the RHS of Eq. (A.2) is negligible and hence Lemma A.3 follows. □

A.2 Lemmas for the proof of Lemma A.3

Lemma A.4. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, and* SPS *is a splitable signature scheme satisfying '$\text{VK}_{\text{SPS-REJ}}$ indistinguishability' as defined in Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. To establish Lemma A.4, we introduce $t^{*(\nu)} + 1$ intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,0}$ and $\mathsf{Hyb}_{0,\nu-1,1}$, namely, $\mathsf{Hyb}_{0,\nu-1,0,\gamma}$, for $\gamma \in [0, t^{*(\nu)}]$ such that $\mathsf{Hyb}_{0,\nu-1,0,t^{*(\nu)}}$ coincides with $\mathsf{Hyb}_{0,\nu-1,0}$ and $\mathsf{Hyb}_{0,\nu-1,0,0}$ coincides with $\mathsf{hyb}_{0,\nu-1,1}$.

Sequence of Intermediate Hybrids Between $\mathsf{Hyb}_{0,\nu-1,0}$ and $\mathsf{Hyb}_{0,\nu-1,1}$

$\mathsf{Hyb}_{0,\nu-1,0,\gamma}$ ($\gamma = 0, \ldots, t^{*(\nu)}$): In this experiment in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first picks PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, \underline{K_{\text{SPS},B}^{(\nu)}}, K_{\text{SPS},E}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next, it generates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
3. It provides $\mathcal{A}$ with the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(0,\gamma)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, \\ \quad \underline{K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*}]) \end{array} \right),$$

   where the program $\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(0,\gamma)}$, depicted in Fig. A.14, is a modification of the program $\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}$, shown in Fig. A.2.

The rest of the experiment is identical to $\mathsf{Hyb}_{0,\nu-1,0}$.

Analysis

Let us denote by $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0,\gamma)}(\lambda)$ the advantage of $\mathcal{A}$, i.e., the absolute difference between 1/2 and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in the hybrid experiment $\mathsf{Hyb}_{0,\nu-1,0,\gamma}$, for $\gamma \in [0, t^{*(\nu)}]$. Clearly, $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0,t^{*(\nu)})}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0,0)}(\lambda)$. Hence, we have

$$|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1)}(\lambda)| \leq \sum_{\gamma=1}^{t^{*(\nu)}} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0,\gamma)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,0,\gamma-1)}(\lambda)|. \tag{A.3}$$

Claim A.1 below justifies that the RHS of Eq. (A.3) is negligible and consequently Lemma A.4 follows.

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}, K_{\text{SPS},B}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   (b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},B})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \ \wedge \ [(t > t^*) \vee (t \leq \gamma) \ \vee \ (h \neq h^*) \ \vee \ (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \ \wedge \ [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \ \wedge \ [\alpha =$'$B$'$]$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, output $\mathcal{F}(K, (h, \ell_{\text{INP}}))$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.14. $\mathsf{Constrained\text{-}Key.Prog}^{(0,\gamma)}_{\text{CPRF}}$

Claim A.1. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, and* SPS *is a splitable signature scheme satisfying '*$\text{VK}_{\text{SPS-REJ}}$ *indistinguishability' as defined in Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,0,\gamma)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,0,\gamma-1)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of Claim A.1 is similar to that of Claim B.1 of [DKW16]. □

□

Lemma A.5. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, and* SPS *is a splitable signature scheme satisfying '*$\text{VK}_{\text{SPS-REJ}}$ *indistinguishability' as defined in Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,2)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. To prove Lemma A.5, we consider the following sequence of intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,1}$ and $\mathsf{Hyb}_{0,\nu-1,2}$:

Sequence of Intermediate Hybrids between $\mathsf{Hyb}_{0,\nu-1,1}$ and $\mathsf{Hyb}_{0,\nu-1,2}$

$\mathbf{Hyb_{0,\nu-1,1,0}}$: This experiment coincides with $\mathsf{Hyb}_{0,\nu-1,1}$.

$\mathbf{Hyb_{0,\nu-1,1,1}}$: In this experiment, to answer the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ selects an additional PPRF key $K^{(\nu)}_{\text{SPS},F} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$ along with all the other PPRF keys as well as the public parameters for positional accumulator and iterator as generated in $\mathsf{Hyb}_{0,\nu-1,1,0}$, providing $\mathcal{A}$ with the constrained key

$$\mathrm{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,1)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, \\ \quad K^{(\nu)}_{\text{SPS},B}, h^*, \ell^*]) \end{array}\right),$$

where the programs $\mathsf{Accumulate.Prog}^{(0,1)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(0,1)}$ are the alterations of the programs $\mathsf{Accumulate.Prog}^{(1)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(1)}$ (Figs. A.3 and A.4) and are depicted in Figs. A.15 and A.16 respectively. The rest of the experiment proceeds in the same way as in $\mathsf{Hyb}_{0,\nu-1,1,0}$.

$\mathbf{Hyb_{0,\nu-1,1,2}}$: In this experiment, in response to the ν^{th} constrained key query of $\mathcal{A}$ corre-

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h,i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h,i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) If $(h,i) = (h^*, \ell^*)$, set $\text{VK} = \text{VK}_{\text{SPS-REJ},F}$.
(d) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
(e) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(f) If $[\alpha =$'-'$] \wedge [(i \neq \ell^*) \vee (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0]$, output $\perp$.
Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(g) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4.(a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
(b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.15. $\mathsf{Accumulate.Prog}^{(0,1)}$

sponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator just as in $\mathsf{Hyb}_{0,\nu-1,1,1}$.

Constants: PPRF keys $K_{\text{SPS},A}, K_{\text{SPS},B}, K_{\text{SPS},E}, K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) If $(h, \ell_{\text{INP}}) = (h^*, \ell^*)$, set $\text{VK} = \text{VK}_{\text{SPS-REJ},F}$.
(d) Set $m = (v, \text{ST}, w, 0)$ and α='-'.
(e) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set α ='E'.
(f) If $[\alpha$ ='-'$] \ \wedge \ [\ell_{\text{INP}} \neq \ell^*) \ \vee \ (h \neq h^*)]$, output $\perp$.
Else if $[\alpha$ ='-'$] \ \wedge \ [\mathsf{SPS.Verify}(\text{VK}, m, \sigma_{\text{SPS,IN}}) = 0]$, output $\perp$.
Else if $[\alpha$ ='-'$] \ \wedge \ [\mathsf{SPS.Verify}(\text{VK}, m, \sigma_{\text{SPS,IN}}) = 1]$, set α ='F'.
(g) If α ='-', output $\perp$.
2.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
(b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{DSPS},B})$.
(c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \ \wedge \ [\alpha$ ='F'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.16. $\mathsf{Change\text{-}SPS.Prog}^{(0,1)}$

2. Next, it forms the punctured PPRF key $K^{(\nu)}_{\text{SPS},F}\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},F}, (h^*, \ell^*))$ as well as computes $r^{(\nu,\ell^*)}_{\text{SPS},H} = \mathcal{F}(K^{(\nu)}_{\text{SPS},F}, (h^*, \ell^*))$ and $(\text{SK}^{(\nu,\ell^*)}_{\text{SPS},H}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS},H}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}) = \mathsf{SPS.Setup}(1^\lambda; r^{(\nu,\ell^*)}_{\text{SPS},H})$.
3. It hands $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}\{(h^*, \ell^*)\}, \\ \quad \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}\{(h^*, \ell^*)\}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, \\ \quad K^{(\nu)}_{\text{SPS},B}, h^*, \ell^*]) \end{array}\right),$$

where the programs $\mathsf{Accumulate.Prog}^{(0,2)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(0,2)}$ are the modifications of the programs $\mathsf{Accumulate.Prog}^{(0,1)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(0,1)}$ (Figs. A.15 and A.16) and are described in Figs. A.17 and A.18 respectively.

The remaining part of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,1,1}$.

$\mathsf{Hyb}_{0,\nu-1,1,3}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,1,2}$ except that while creating the ν^{th} constrained key queried by $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ selects $r^{(\nu,\ell^*)}_{\text{SPS},H} \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, i.e., in other words, $\mathcal{B}$ generates $(\text{SK}^{(\nu,\ell^*)}_{\text{SPS},H}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS},H}, \text{VK}^{(\nu,\ell^*)}_{\text{SPS-REJ},H}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$,

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF key $K_{\text{SPS},E}$, Punctured PPRF key $K_{\text{SPS},F}\{(h^*, \ell^*)\}$, Verification key VK_H, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \ell^*)\}, (h, i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =$'-'$] \wedge [(i \neq \ell^*) \vee (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =$'-'$] \wedge \underline{[\mathsf{SPS.Verify}(\text{VK}_H, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0]}$, output $\perp$.
Else if $[\alpha =$'-'$] \wedge \underline{[\mathsf{SPS.Verify}(\text{VK}_H, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]}$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4.(a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
(b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.17. $\mathsf{Accumulate.Prog}^{(0,2)}$

Constants: PPRF keys $K_{\text{SPS},A}, K_{\text{SPS},B}, K_{\text{SPS},E}$, Punctured PPRF key $K_{\text{SPS},F}\{(h^*, \ell^*)\}$, Verification key VK_H, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \ell^*)\}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m = (v, \text{ST}, w, 0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =$'-'$] \wedge [(\ell_{\text{INP}} \neq \ell^*) \vee (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =$'-'$] \wedge \underline{[\mathsf{SPS.Verify}(\text{VK}_H, m, \sigma_{\text{SPS,IN}}) = 0]}$, output $\perp$.
Else if $[\alpha =$'-'$] \wedge \underline{[\mathsf{SPS.Verify}(\text{VK}_H, m, \sigma_{\text{SPS,IN}}) = 1]}$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
(b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{DSPS},B})$.
(c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [\alpha =$'$F$'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.18. $\mathsf{Change\text{-}SPS.Prog}^{(0,2)}$

and gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \ell^*)\}, \\ \quad \text{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \ell^*)\}, \text{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, \\ \quad K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

$\mathsf{Hyb}_{0,\nu-1,1,4}$: In this experiment, to answer the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ creates all the components as in $\mathsf{Hyb}_{0,\nu-1,1,3}$, however, it provides $\mathcal{A}$ with the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \ell^*)\}, \\ \quad \underline{\mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\ell^*)}}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \ell^*)\}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\ell^*)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, \overline{K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}}, K_{\mathrm{SPS},A}^{(\nu)}, \\ \quad K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

The rest of the experiment is the same as $\mathsf{Hyb}_{0,\nu-1,1,3}$.

$\mathsf{Hyb}_{0,\nu-1,1,5}$: In this experiment, in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ forms all the components as in $\mathsf{Hyb}_{0,\nu-1,1,4}$ except that it computes $\underline{r_{\mathrm{SPS},H}^{(\nu,\ell^*)} = \mathcal{F}(K_{\mathrm{SPS},F}^{(\nu)}, (h^*, \ell^*)), (\mathrm{SK}_{\mathrm{SPS},H}^{(\nu,\ell^*)}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\ell^*)}, \mathrm{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\mathrm{SPS},H}^{(\nu,\ell^*)})}$, and hands $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \ell^*)\}, \\ \quad \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\ell^*)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \ell^*)\}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\ell^*)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, \\ \quad K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,1,4}$.

$\mathsf{Hyb}_{0,\nu-1,1,6}$: This experiment corresponds to $\mathsf{Hyb}_{0,\nu-1,2}$.

Analysis

Let $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,\vartheta)}(\lambda)$ represents the advantage of $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in $\mathsf{Hyb}_{0,\nu-1,1,\vartheta}$, for $\vartheta \in [0, 6]$. By definition, $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,0)}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,6)}(\lambda)$. Then, we have

$$|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2)}(\lambda)| \leq \sum_{\vartheta=1}^{6} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,\vartheta-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,\vartheta)}(\lambda)|. \tag{A.4}$$

Claims A.2–A.7 below will demonstrate that the RHS of Eq. (A.4) is negligible and thus Lemma A.5 follows.

Claim A.2. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,0)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,1)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The difference between $\mathsf{Hyb}_{0,\nu-1,1,0}$ and $\mathsf{Hyb}_{0,\nu-1,1,1}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,1,0}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_0)$ and $\mathcal{IO}(P_0')$ within the ν^{th} constrained key returned to $\mathcal{A}$, while in $\mathsf{Hyb}_{0,\nu-1,1,1}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_1)$ and $\mathcal{IO}(P_1')$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]$ (Fig. 3.2),
- $P_0' = \mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]$ (Fig. 3.3),
- $P_1 = \mathsf{Accumulate.Prog}^{(0,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]$ (Fig. A.15),
- $P_1' = \mathsf{Change\text{-}SPS.Prog}^{(0,1)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]$ (Fig. A.16).

Now, observe that the programs P_0 and P_1 clearly have identical outputs for inputs corresponding to $(h, i) \neq (h^*, \ell^*)$. Also, by the correctness [Property (vii)] of splitable signature scheme SPS, both the programs output $\perp$ in case $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$ for inputs corresponding to (h^*, ℓ^*). Thus the programs P_0 and P_1 are functionally equivalent. A similar argument justifies the functional equivalence of the programs P_0' and P_1'.

Thus, by the security of $\mathcal{IO}$, Claim A.2 follows. Ofcourse, we need to consider a sequence of hybrid experiments to arrive at the result where in each hybrid experiment we change the programs one at a time. □

Claim A.3. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for $\mathsf{P/poly}$ and $\mathcal{F}$ satisfy the correctness under puncturing property defined in Definition 2.2, for any PPT adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,2)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function negl.*

Proof. The difference between $\mathsf{Hyb}_{0,\nu-1,1,1}$ and $\mathsf{Hyb}_{0,\nu-1,1,2}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,1,1}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_0)$ and $\mathcal{IO}(P_0')$ within the ν^{th} constrained key returned to $\mathcal{A}$, while in $\mathsf{Hyb}_{0,\nu-1,1,2}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_1)$ and $\mathcal{IO}(P_1')$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(0,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]$ (Fig. A.15),
- $P_0' = \mathsf{Change\text{-}SPS.Prog}^{(0,1)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]$ (Fig. A.16),
- $P_1 = \mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \ell^*)\}, \text{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}, h^*, \ell^*]$ (Fig. A.17),
- $P_1' = \mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \ell^*)\}, \text{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}, h^*, \ell^*]$ (Fig. A.18).

Now, by the correctness under puncturing property of the PPRF $\mathcal{F}$, both the programs P_0 and P_1 have identical outputs on inputs corresponding to $(h, i) \neq (h^*, \ell^*)$. For inputs corresponding to (h^*, ℓ^*), P_1 uses the hardwired verification key $\text{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}$, where in $\mathsf{Hyb}_{0,\nu-1,1,2}$, $\text{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}$ is computed as $(\text{SK}_{\text{SPS},H}^{(\nu,\ell^*)}, \text{VK}_{\text{SPS},H}^{(\nu,\ell^*)}, \text{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},H}^{(\nu,\ell^*)})$ and $r_{\text{SPS},H}^{(\nu,\ell^*)} = \mathcal{F}(K_{\text{SPS},F}^{(\nu)}, (h^*, \ell^*))$. Observe that these values are exactly the same as those used by the program P_0 for inputs corresponding to (h^*, ℓ^*). Thus, both programs have identical outputs for inputs corresponding to (h^*, ℓ^*) as well. Hence, the two programs are functionally equivalent. A similar argument will justify that the programs P_0' and P_1' are functionally equivalent.

Therefore, by the security of $\mathcal{IO}$, Claim A.3 follows, considering a sequence of hybrid experiments where in each hybrid experiment we change the programs one at a time. □

Claim A.4. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any PPT adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,2)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,3)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function negl.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,2)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,3)}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $mathcal A$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. On the other hand, if $b = 1$, then it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
  4. $\mathcal{B}$ returns the challenge CPRF value y^* to $\mathcal{A}$.
- For $\eta \in [\hat{q}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,1,2}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it creates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. Then, $\mathcal{B}$ sends (h^*, ℓ^*) as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back a punctured PPRF key $K^*\{(h^*, \ell^*)\}$ and a value $r^* \in \mathcal{Y}_{\text{PPRF}}$, where either $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$ or $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$. $\mathcal{B}$ will *implicitly* view the key K^* as the key $K_{\text{SPS},F}^{(\nu)}$.
  4. $\mathcal{B}$ generates $(\text{SK}_{\text{SPS},H}^{(\nu,\ell^*)}, \text{VK}_{\text{SPS},H}^{(\nu,\ell^*)}, \text{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}) = \mathsf{SPS.Setup}(1^\lambda;\ r^*)$.
  5. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K^*\{(h^*, \ell^*)\}, \\ \qquad \text{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K^*\{(h^*, \ell^*)\}, \text{VK}_{\text{SPS-REJ},H}^{(\nu,\ell^*)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \qquad K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its PPRF selective pseudorandomness experiment.

Note that if $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,1,2}$. On the other hand, if $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,1,3}$. This completes the proof of Claim A.4. □

Claim A.5. *Assuming* SPS *is a splitable signature scheme satisfying* '$\text{VK}_{\text{SPS-REJ}}$ *indistinguishability' as per Definition 2.6, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,4)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,4)}(\lambda)|$ is non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the $\text{VK}_{\text{SPS-REJ}}$ indistinguishability of SPS using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ receives a verification key VK of the splitable signature scheme SPS from its $\text{VK}_{\text{SPS-REJ}}$ indistinguishability challenger $\mathcal{C}$, where VK is either a proper verification key VK_{SPS} or a reject verification key $\text{VK}_{\text{SPS-REJ}}$. Then, $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:

1. $\mathcal{B}$ generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
3. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. On the other hand, if $b = 1$, then it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
4. $\mathcal{B}$ returns the challenge CPRF value y^* to $\mathcal{A}$.

- For $\eta \in [\hat{q}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,1,3}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it creates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. $\mathcal{B}$ creates the punctured PPRF key $K_{\text{SPS},F}^{(\nu)}\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},F}, (h^*, \ell^*))$.
  4. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(0,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \ell^*)\}, \\ \quad \text{VK}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(0,2)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \ell^*)\}, \text{VK}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its SPS $\text{VK}_{\text{SPS-REJ}}$ indistinguishability experiment.

Notice that if $\text{VK} = \text{VK}_{\text{SPS-REJ}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,1,3}$. On the other hand, if $\text{VK} = \text{VK}_{\text{SPS}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,1,4}$. This completes the proof of Claim A.5. □

Claim A.6. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,4)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,5)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of Claim A.6 is similar to that of Claim A.4 with some appropriate changes which can be readily identified. □

Claim A.7. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$ satisfies the correctness under puncturing property defined in Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,5)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,1,6)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of Claim A.7 is analogous to that of Claim A.3 with some appropriate changes that are easy to determine. □

□

Lemma A.6. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, and* SPS *is a splitable signature scheme satisfying '$\text{VK}_{\text{SPS-REJ}}$ indistinguishability' defined in Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. To prove Lemma A.6, we consider the following sequence of ℓ^* intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,2}$ and $\mathsf{Hyb}_{0,\nu-1,3}$:

Sequence of Intermediate Hybrids between $\mathsf{Hyb}_{0,\nu-1,2}$ and $\mathsf{Hyb}_{0,\nu-1,3}$

$\mathsf{Hyb}_{0,\nu-1,2,\iota}$ $(\iota = 0, \ldots, \ell^* - 1)$: In this experiment in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first chooses PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next, it generates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
3. It provides $\mathcal{A}$ with the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\underline{\mathsf{Accumulate.Prog}^{(1,\iota)}}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\underline{\mathsf{Change\text{-}SPS.Prog}^{(1,\iota)}}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)} \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right),$$

where the programs $\mathsf{Accumulate.Prog}^{(1,\iota)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(1,\iota)}$ are the modifications of the programs $\mathsf{Accumulate.Prog}^{(2)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(2)}$ (Figs. A.5 and A.6) and are depicted in Figs. A.19 and A.20 respectively.

The rest of the experiment is similar to $\mathsf{Hyb}_{0,\nu-1,2}$. Observe that $\mathsf{Hyb}_{0,\nu-1,2,\ell^*-1}$ coincides with $\mathsf{hyb}_{0,\nu-1,2}$ and $\mathsf{Hyb}_{0,\nu-1,2,0}$ corresponds to $\mathsf{hyb}_{0,\nu-1,3}$.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =$'-'$] \wedge [(i > \ell^*) \vee \underline{(i \leq \iota)} \vee (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4.(a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
(b) Compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i+1))$, $(\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},F})$.
(c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. If $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.19. $\mathsf{Accumulate.Prog}^{(1,\iota)}$

Constants: PPRF keys $K_{\text{SPS},A}$, $K_{\text{SPS},B}$, $K_{\text{SPS},E}$, $K_{\text{SPS},F}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m = (v, \text{ST}, w, 0)$ and α='-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set α ='E'.
(e) If $[\alpha$ ='-'$] \wedge [(\ell_{\text{INP}} > \ell^*) \vee \underline{(\ell_{\text{INP}} \leq \iota)} \vee (h \neq h^*)]$, output $\perp$.
Else if $[\alpha$ ='-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m, \sigma_{\text{SPS,IN}}) = 1]$, set α ='F'.
(f) If α ='-', output $\perp$.
2.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
(b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},B})$.
(c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [\alpha$ ='F'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.20. Change-SPS.Prog$^{(1,\iota)}$

Analysis

Let us denote by $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,\iota)}(\lambda)$ the advantage of $\mathcal{A}$, i.e., the absolute difference between 1/2 and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in the hybrid experiment $\mathsf{Hyb}_{0,\nu-1,2,\iota}$, for $\iota \in [0, \ell^* - 1]$. Clearly, $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,\ell^*-1)}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,0)}(\lambda)$. Hence we have,

$$|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3)}(\lambda)| \leq \sum_{\iota=1}^{\ell^*-1} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,\iota)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,\iota-1)}(\lambda)|. \tag{A.5}$$

Claim A.8 below justifies that the RHS of Eq. (A.5) is negligible and consequently Lemma A.6 follows.

Claim A.8. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, and* SPS *is a splitable signature scheme satisfying* '$\text{VK}_{\text{SPS-REJ}}$ *indistinguishability' as defined in Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,\iota)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,2,\iota-1)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim A.8 is similar to that of Lemma A.5 with some appropriate modifications which are easy to find out. □
□

Lemma A.7. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, and* SPS *is a splittable signature scheme satisfying* '$\text{VK}_{\text{SPS-ONE}}$ *indistinguishability' as defined in Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,0)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. In order to prove Lemma A.7, we consider the following sequence of intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,3}$ and $\mathsf{Hyb}_{0,\nu-1,3,0}$.

Sequence of Intermediate Hybrids between $\mathsf{Hyb}_{0,\nu-1,3}$ and $\mathsf{Hyb}_{0,\nu-1,3,0}$

$\mathsf{Hyb}_{0,\nu-1,3\text{-I}}$: This experiment coincides with $\mathsf{Hyb}_{0,\nu-1,3}$.

$\mathsf{Hyb}_{0,\nu-1,3\text{-II}}$: In this experiment, to answer the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys together with the public parameters for the positional accumulator and the iterator just as in $\mathsf{hyb}_{0,\nu-1,3\text{-I}}$.
2. Next, it forms the punctured PPRF key $K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},E},(h^*,0))$ as well as computes $r^{(\nu,0)}_{\text{SPS},G} = \mathcal{F}(K^{(\nu)}_{\text{SPS},E},(h^*,0))$ and $(\text{SK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},G} = \mathsf{SPS.Setup}(1^\lambda; r^{(\nu,0)}_{\text{SPS},G})$.
3. Then, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$ and computes $\sigma^{(\nu,0)}_{\text{SPS},G} = \mathsf{SPS.Sign}(\text{SK}^{(\nu,0)}_{\text{SPS},G}, m^{(\nu)}_{0,0})$.
4. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, \sigma^{(\nu,0)}_{\text{SPS},G}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, K^{(\nu)}_{\text{SPS},F}, \text{VK}^{(\nu,0)}_{\text{SPS},G}, \\ \quad h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\}, K^{(\nu)}_{\text{SPS},F}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, \\ \quad K^{(\nu)}_{\text{SPS},B}, h^*, \ell^*]) \end{array}\right),$$

where the programs $\mathsf{Init\text{-}SPS.Prog}^{(1)}$ and $\mathsf{Accumulate.Prog}^{(2,1)}$ respectively are the alterations of the programs Init-SPS.Prog and $\mathsf{Accumulate.Prog}^{(2)}$ (Figs. 3.1 and A.5) and are depicted in Figs. A.21 and A.22.

The remaining part of the experiment is similar to $\mathsf{Hyb}_{0,\nu-1,3\text{-I}}$.

$\mathsf{hyb}_{0,\nu-1,3\text{-III}}$: This experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,3\text{-II}}$ with the only exception that while

Constants: Initial TM state q_0, Accumulator value w_0, Iterator value v_0, Punctured PPRF key $K_{\text{SPS},E}\{(h^*,0)\}$, Signature σ_G, SSB hash value of challenge input h^*

Input: SSB hash value h

Output: Signature $\sigma_{\text{SPS,OUT}}$

1. If $h = h^*$, output σ_G.
   Else, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,0)\},(h,0))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
2. Output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},E}, (v_0, q_0, w_0, 0))$.

Fig. A.21. $\mathsf{Init\text{-}SPS.Prog}^{(1)}$

constructing the ν^{th} constrained key queried by $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ selects $r^{(\nu,0)}_{\text{SPS},G} \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$. More formally, to answer the ν^{th} constrained key query of $\mathcal{A}$, $\mathcal{B}$ creates all the components as in $\mathsf{Hyb}_{0,\nu-1,3\text{-II}}$ except that it generates $(\text{SK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS},G}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},G}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$, sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$, computes $\sigma^{(\nu,0)}_{\text{SPS},G} = \mathsf{SPS.Sign}(\text{SK}^{(\nu,0)}_{\text{SPS},G},$

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, Punctured PPRF key $K_{\text{SPS},E}\{(h^*,0)\}$, PPRF key $K_{\text{SPS},F}$, Verification key VK_G, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) If $(h,i) \neq (h^*,0)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,0)\},(h,i))$, $(\text{SK}_{\text{SPS},E},\text{VK}_{\text{SPS},E},\text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F},(h,i))$, $(\text{SK}_{\text{SPS},F},\text{VK}_{\text{SPS},F},\text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m_{\text{IN}} = (v_{\text{IN}},\text{ST},w_{\text{IN}},0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E},m_{\text{IN}},\sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =\text{'-'}] \wedge [(i > \ell^*) \vee (i = 0) \vee (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F},m_{\text{IN}},\sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK},h,i,\text{SYM}_{\text{IN}},\pi_{\text{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}},w_{\text{IN}},\text{SYM}_{\text{IN}},i,\text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}},v_{\text{IN}},(\text{ST},w_{\text{IN}},0))$.
4.(a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,0)\},(h,i+1))$, $(\text{SK}'_{\text{SPS},E},\text{VK}'_{\text{SPS},E},\text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
(b) Compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F},(h,i+1))$, $(\text{SK}'_{\text{SPS},F},\text{VK}'_{\text{SPS},F},\text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},F})$.
(c) Set $m_{\text{OUT}} = (v_{\text{OUT}},\text{ST},w_{\text{OUT}},0)$. If $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha},m_{\text{OUT}})$.
Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E},m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}},v_{\text{OUT}},\sigma_{\text{SPS,OUT}})$.

Fig. A.22. $\mathsf{Accumulate.Prog}^{(2,1)}$

$m^{(\nu)}_{0,0}$) and provides $\mathcal{A}$ with the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK},\text{PP}^{(\nu)}_{\text{ACC}},w^{(\nu)}_0,\text{STORE}^{(\nu)}_0,\text{PP}^{(\nu)}_{\text{ITR}},v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q^{(\nu)}_0,w^{(\nu)}_0,v^{(\nu)}_0,K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\},\sigma^{(\nu,0)}_{\text{SPS},G},h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}}=2^\lambda,\text{HK},\text{PP}^{(\nu)}_{\text{ACC}},\text{PP}^{(\nu)}_{\text{ITR}},K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\},K^{(\nu)}_{\text{SPS},F},\text{VK}^{(\nu,0)}_{\text{SPS},G}, \\ \quad h^*,\ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K^{(\nu)}_{\text{SPS},A},K^{(\nu)}_{\text{SPS},B},K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\},K^{(\nu)}_{\text{SPS},F},h^*,\ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{CPRF}}[M^{(\nu)},T=2^\lambda,t^{*(\nu)},\text{PP}^{(\nu)}_{\text{ACC}},\text{PP}^{(\nu)}_{\text{ITR}},K,K^{(\nu)}_1,\ldots,K^{(\nu)}_\lambda,K^{(\nu)}_{\text{SPS},A}, \\ \quad K^{(\nu)}_{\text{SPS},B},h^*,\ell^*]) \end{array}\right).$$

$\mathsf{Hyb}_{0,\nu-1,3\text{-IV}}$: This experiment is the same as $\mathsf{hyb}_{0,\nu-1,3\text{-III}}$ with the exception that in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds sa follows:

1. It first generates the full and punctured PPRF keys together with the public parameters for the positional accumulator and the iterator just as in $\mathsf{Hyb}_{0,\nu-1,3\text{-III}}$.
2. Next, it creates $(\text{SK}^{(\nu,0)}_{\text{SPS},G},\text{VK}^{(\nu,0)}_{\text{SPS},G},\text{VK}^{(\nu,0)}_{\text{SPS-REJ},G}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$, sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0,q^{(\nu)}_0,w^{(\nu)}_0,0)$, and forms $(\sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{0,0},G},\text{VK}^{(\nu,0)}_{\text{SPS-ONE},G},\text{SK}^{(\nu,0)}_{\text{SPS-ABO},G},\text{VK}^{(\nu,0)}_{\text{SPS-ABO},G}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}^{(\nu,0)}_{\text{SPS},G},m^{(\nu)}_{0,0})$.

3. $\mathcal{B}$ hands $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, \underline{\sigma^{(\nu,0)}_{\text{SPS-ONE},m_{0,0}^{(\nu)},G}}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \underline{\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, \\ \quad \underline{\mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},G}}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

$\mathsf{Hyb}_{0,\nu-1,3\text{-V}}$: In this experiment, in reply to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the components just as in $\mathsf{Hyb}_{0,\nu-1,3\text{-IV}}$ and gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, \sigma^{(\nu,0)}_{\text{SPS-ONE},m_{0,0}^{(\nu)},G}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, \\ \quad \underline{\mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},G}, m_{0,0}^{(\nu)}, h^*, \ell^*]}), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right),$$

where the program $\mathsf{Accumulate.Prog}^{(2,2)}$, described in Fig. A.23, is an alteration of the program $\mathsf{Accumulate.Prog}^{(2,1)}$ (Fig. A.22). The rest of the experiment is similar to $\mathsf{Hyb}_{0,\nu-1,3\text{-IV}}$.

$\mathsf{Hyb}_{0,\nu-1,3\text{-VI}}$: In this experiment, in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the full and punctured PPRF keys as well as the public parameters for the positional accumulator and its iterator as in $\mathsf{Hyb}_{0,\nu-1,3\text{-V}}$.
2. Next, it forms $(\mathrm{SK}_{\mathrm{SPS},G}^{(\nu,0)}, \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,0)}, \mathrm{VK}_{\text{SPS-REJ},G}^{(\nu,0)}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$, sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$, and computes $\underline{\sigma_{\mathrm{SPS},G}^{(\nu,0)} = \mathsf{SPS.Sign}(\mathrm{SK}_{\mathrm{SPS},G}^{(\nu,0)}, m_{0,0}^{(\nu)})}$.
3. It provides $\mathcal{A}$ with the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, \underline{\sigma^{(\nu,0)}_{\mathrm{SPS},G}}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \underline{\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, \\ \quad \underline{\mathrm{VK}^{(\nu,0)}_{\mathrm{SPS},G}}, m_{0,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, Punctured PPRF key $K_{\text{SPS},E}\{(h^*,0)\}$, PPRF key $K_{\text{SPS},F}$, Verification key VK_G, Message $m_{0,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) If $(h,i) \neq (h^*,0)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,0)\},(h,i))$, $(\text{SK}_{\text{SPS},E},\text{VK}_{\text{SPS},E},\text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F},(h,i))$, $(\text{SK}_{\text{SPS},F},\text{VK}_{\text{SPS},F},\text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m_{\text{IN}} = (v_{\text{IN}},\text{ST},w_{\text{IN}},0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E},m_{\text{IN}},\sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =\text{'-'}] \wedge [(i > \ell^*) \vee (i = 0) \vee (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F},m_{\text{IN}},\sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK},h,i,\text{SYM}_{\text{IN}},\pi_{\text{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}},w_{\text{IN}},\text{SYM}_{\text{IN}},i,\text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}},v_{\text{IN}},(\text{ST},w_{\text{IN}},0))$.
4.(a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,0)\},(h,i+1))$, $(\text{SK}'_{\text{SPS},E},\text{VK}'_{\text{SPS},E},\text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
(b) Compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F},(h,i+1))$, $(\text{SK}'_{\text{SPS},F},\text{VK}'_{\text{SPS},F},\text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},F})$.
(c) Set $m_{\text{OUT}} = (v_{\text{OUT}},\text{ST},w_{\text{OUT}},0)$.
If $[(h,i) = (h^*,0)] \wedge [m_{\text{IN}} = m_{0,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E},m_{\text{OUT}})$.
Else if $[(h,i) = (h^*,0)] \wedge [m_{\text{IN}} \neq m_{0,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F},m_{\text{OUT}})$.
Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha},m_{\text{OUT}})$.
Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}},v_{\text{OUT}},\sigma_{\text{SPS,OUT}})$.

Fig. A.23. $\mathsf{Accumulate.Prog}^{(2,2)}$

The remaining portion of the experiment is identical to $\mathsf{hyb}_{0,\nu-1,3\text{-V}}$.

$\mathsf{hyb}_{0,\nu-1,3\text{-VII}}$: In this experiment, while constructing the ν^{th} constrained key queried by $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates everything just as in $\mathsf{Hyb}_{0,\nu-1,3\text{-VI}}$ except that it computes $r^{(\nu,0)}_{\text{SPS},G} = \mathcal{F}(K^{(\nu)}_{\text{SPS},E},(h^*,0))$, forms $(\text{SK}^{(\nu,0)}_{\text{SPS},G},\text{VK}^{(\nu,0)}_{\text{SPS},G},\text{VK}^{(\nu,0)}_{\text{SPS-REJ},G}) = \mathsf{SPS.Setup}(1^\lambda;\ r^{(\nu,0)}_{\text{SPS},G})$, and provides $\mathcal{A}$ with the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK},\text{PP}^{(\nu)}_{\text{ACC}},w^{(\nu)}_0,\text{STORE}^{(\nu)}_0,\text{PP}^{(\nu)}_{\text{ITR}},v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q^{(\nu)}_0,w^{(\nu)}_0,v^{(\nu)}_0,K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\},\sigma^{(\nu,0)}_{\text{SPS},G},h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,2)}[n_{\text{SSB-BLK}} = 2^\lambda,\text{HK},\text{PP}^{(\nu)}_{\text{ACC}},\text{PP}^{(\nu)}_{\text{ITR}},K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\},K^{(\nu)}_{\text{SPS},F}, \\ \qquad \text{VK}^{(\nu,0)}_{\text{SPS},G},m^{(\nu)}_{0,0},h^*,\ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K^{(\nu)}_{\text{SPS},A},K^{(\nu)}_{\text{SPS},B},K^{(\nu)}_{\text{SPS},E}\{(h^*,0)\},K^{(\nu)}_{\text{SPS},F},h^*,\ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{CPRF}}[M^{(\nu)},T = 2^\lambda,t^{*(\nu)},\text{PP}^{(\nu)}_{\text{ACC}},\text{PP}^{(\nu)}_{\text{ITR}},K,K^{(\nu)}_1,\ldots,K^{(\nu)}_\lambda, \\ \qquad K^{(\nu)}_{\text{SPS},A},K^{(\nu)}_{\text{SPS},B},h^*,\ell^*]) \end{array}\right).$$

The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,3\text{-VI}}$.

$\mathsf{Hyb}_{0,\nu-1,3\text{-VIII}}$: This experiment corresponds to $\mathsf{Hyb}_{0,\nu-1,3,0}$.

Analysis

Let $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}\vartheta)}(\lambda)$ represents the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in $\mathsf{Hyb}_{0,\nu-1,3\text{-}\vartheta}$, for $\vartheta \in \{\mathrm{I},\ldots,\mathrm{VIII}\}$. Clearly, $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-I})}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,0)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-VIII})}(\lambda)$. Therefore, we have

$$|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,0)}(\lambda)| \leq \sum_{\vartheta=\mathrm{II}}^{\mathrm{VIII}} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}(\vartheta-\mathrm{I}))}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}\vartheta)}(\lambda)|. \tag{A.6}$$

Claims A.9–A.15 below will justify that the RHS of Eq. (A.6) is negligible and hence Lemma A.7 follows.

Claim A.9. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$ *and $\mathcal{F}$ satisfies the correctness under puncturing property defined in Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}I)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}II)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The difference between $\mathsf{Hyb}_{0,\nu-1,3\text{-I}}$ and $\mathsf{hyb}_{0,\nu-1,3\text{-II}}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,3\text{-I}}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_0)$ and $\mathcal{IO}(P_0')$ within the ν^{th} constrained key returned to $\mathcal{A}$, while in $\mathsf{Hyb}_{0,\nu-1,3\text{-II}}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_1)$ and $\mathcal{IO}(p_1')$ instead, where

- $P_0 = \mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]$ (Fig. 3.1),
- $P_0' = \mathsf{Accumulate.Prog}^{(2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]$ (Fig. A.5),
- $P_1 = \mathsf{Init\text{-}SPS.Prog}^{(1)}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, \sigma_{\mathrm{SPS},G}^{(\nu,0)}, h^*]$ (Fig. A.21),
- $P_1' = \mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,0)\}, K_{\mathrm{SPS},F}^{(\nu)}, \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,0)}, h^*, \ell^*]$ (Fig. A.22).

Now observe that the programs P_0 and P_1 are functionally equivalent since by the correctness under puncturing property of the PPRF $\mathcal{F}$, the PPRF output remains the same at all non-punctured points and at the point of puncturing, i.e., $(h^*,0)$, the correct signature is hardwired in the program P_1. Similarly, P_0' and P_1' are also functionally equivalent by the correctness under puncturing property of $\mathcal{F}$ and the fact that at the point of puncturing i.e., $(h^*,0)$ the correct verification key is hardwired into the program P_1'.

Therefore, by the security of $\mathcal{IO}$, Claim A.9 follows. □

Claim A.10. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}II)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}III)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-II})}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-III})}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\mathrm{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. On the other hand, if $b = 1$, then it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$.
  4. $\mathcal{B}$ returns the challenge CPRF value y^* to $\mathcal{A}$.

- For $\eta \in [\hat{q}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3\text{-II}}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it creates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. Then, $\mathcal{B}$ sends $(h^*, 0)$ as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back a punctured PPRF key $K^*\{(h^*, 0)\}$ and a value $r^* \in \mathcal{Y}_{\text{PPRF}}$, where either $r^* = \mathcal{F}(K^*, (h^*, 0))$ or $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$. $\mathcal{B}$ *implicitly* views the key K^* as the key $K_{\text{SPS},E}^{(\nu)}$.
  4. $\mathcal{B}$ generates $(\text{SK}_{\text{SPS},G}^{(\nu,0)}, \text{VK}_{\text{SPS},G}^{(\nu,0)}, \text{VK}_{\text{SPS-REJ},G}^{(\nu,0)}) = \mathsf{SPS.Setup}(1^\lambda;\ r^*)$.
  5. Then, $\mathcal{B}$ sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$ and computes $\sigma_{\text{SPS},G}^{(\nu,0)} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},G}^{(\nu,0)}, m_{0,0}^{(\nu)})$.
  6. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K^*\{(h^*,0)\}, \sigma_{\text{SPS},G}^{(\nu,0)}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K^*\{(h^*,0)\}, K_{\text{SPS},F}^{(\nu)}, \\ \qquad \text{VK}_{\text{SPS},G}^{(\nu,0)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K^*\{(h^*,0)\}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \qquad K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its PPRF selective pseudorandomness experiment.

Note that if $r^* = \mathcal{F}(K^*, (h^*, 0))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3\text{-II}}$. On the other hand, if $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, the $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3\text{-III}}$. This completes the proof of Claim A.10. □

Claim A.11. *Assuming* SPS *is a splitable signature scheme satisfying* '$\text{VK}_{\text{SPS-ONE}}$' *indistinguishability as per Definition 2.6, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}III)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}IV)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-III})}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-IV})}(\lambda)|$ is non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the $\text{VK}_{\text{SPS-ONE}}$ indistinguishability of SPS using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. On the other hand, if $b = 1$, then it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
  4. $\mathcal{B}$ returns the challenge CPRF value y^* to $\mathcal{A}$.
- For $\eta \in [\hat{q}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3\text{-III}}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:

1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
2. Next, it creates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
3. Then, $\mathcal{B}$ creates the punctured PPRF key $K_{\text{SPS},E}^{(\nu)}\{(h^*, 0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},E}^{(\nu)}, (h^*, 0))$.
4. After that, $\mathcal{B}$ sends $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$ as the challenge message to its SPS $\text{VK}_{\text{SPS-ONE}}$ indistinguishability challenger $\mathcal{C}$ and receives back a signature-verification key pair $(\sigma_{\text{SPS-ONE},m_{0,0}^{(\nu)}}, \text{VK})$, where VK is either a normal verification key VK_{SPS} or a one verification key $\text{VK}_{\text{SPS-ONE}}$ for the message $m_{0,0}^{(\nu)}$.
5. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}^{(1)}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, 0)\}, \sigma_{\text{SPS-ONE},m_{0,0}^{(\nu)}}, h^*]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, 0)\}, K_{\text{SPS},F}^{(\nu)}, \\ \qquad \text{VK}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(2)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, 0)\}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \qquad K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0, 1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its SPS $\text{VK}_{\text{SPS-ONE}}$ indistinguishability experiment.

Notice that if $\text{VK} = \text{VK}_{\text{SPS}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3\text{-III}}$. On the other hand, if $\text{VK} = \text{VK}_{\text{SPS-ONE}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3\text{-IV}}$. This completes the proof of Claim A.11. □

Claim A.12. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}IV)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}V)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The difference between $\mathsf{Hyb}_{0,\nu-1,3\text{-IV}}$ and $\mathsf{hyb}_{0,\nu-1,3\text{-V}}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,3\text{-IV}}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0)$ within the ν^{th} constrained key returned to $\mathcal{A}$, while in $\mathsf{Hyb}_{0,\nu-1,3\text{-V}}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_1)$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(2,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, 0)\}, K_{\text{SPS},F}^{(\nu)}, \text{VK}_{\text{SPS-ONE},G}^{(\nu,0)}, h^*, \ell^*]$ (Fig. A.22),
- $P_1 = \mathsf{Accumulate.Prog}^{(2,2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, 0)\}, K_{\text{SPS},F}^{(\nu)}, \text{VK}_{\text{SPS-ONE},G}^{(\nu,0)}, m_{0,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.23).

Observe that the only inputs for which the programs P_0 and P_1 can possibly differ are those corresponding to $(h, i) = (h^*, 0)$. However, the verification key hardwired in both the programs is $\text{VK}_{\text{SPS-ONE},G}^{(\nu,0)}$ which only accepts signature for $m_{\text{IN}} = m_{0,0}^{(\nu)}$ by the correctness [Properties (i), (iii) and (v)]. This ensures that for inputs corresponding to $(h^*, 0)$, if $m_{\text{IN}} = m_{0,0}^{(\nu)}$ both the programs output an 'E' type signature, else, both output $\perp$. Thus, P_0 and P_1 are functionally equivalent.

Therefore, by the security of $\mathcal{IO}$, Claim A.12 follows. □

Claim A.13. *Assuming* SPS *is a splitable signature scheme satisfying '$\text{VK}_{\text{SPS-ONE}}$ indistinguishability' as per Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}V)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}VI)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of Claim A.13 is similar to that of Claim A.11 with some readily identifiable modifications. □

Claim A.14. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}VI)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}VII)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim A.14 is analogous to that of Claim A.10 with some appropriate changes that are easy to find out. □

Claim A.15. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$ satisfies the correctness under puncturing property defined in Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}VII)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3\text{-}VIII)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim A.15 is analogous to that of Claim A.9. □

□

Lemma A.8. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly, SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a positional accumulator satisfying 'indistinguishability of write setup' and 'write enforcing' properties defined in Definition 2.4, as well as* ITR *is a secure cryptographic iterator according to Definition 2.5, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota')}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. To prove Lemma A.8, we introduce the following sequence of intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,3,\iota}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota'}$:

Sequence of Intermediate Hybrids between $\mathbf{Hyb}_{0,\nu-1,3,\iota}$ and $\mathbf{Hyb}_{0,\nu-1,3,\iota'}$

$\mathbf{Hyb}_{0,\nu-1,3,\iota,0}$: This experiment coincides with $\mathsf{Hyb}_{0,\nu-1,3,\iota}$.

$\mathbf{Hyb}_{0,\nu-1,3,\iota,1}$: In this experiment the challenger $\mathcal{B}$ generates the SSB hash key $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = \iota)$. The rest of the experiment proceeds in an analogous fashion to $\mathsf{Hyb}_{0,\nu-1,3,\iota,0}$.

$\mathbf{Hyb}_{0,\nu-1,3,\iota,2}$: In this experiment, to answer the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys and the public parameters for the iterator just as in $\mathsf{hyb}_{0,\nu-1,3,\iota,1}$.
2. After that, it generates $(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x_0^*, 0), \ldots, (x_\iota^*, \iota)))$.
3. Next, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $j = 1, \ldots, \iota$, it iteratively computes the following:
   - $\mathrm{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1)$
   - $w_j^{(\nu)} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \mathrm{AUX}_j^{(\nu)})$
   - $\mathrm{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
   - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$

   It sets $m_{\iota,0}^{(\nu)} = (v_\iota^{(\nu)}, q_0^{(\nu)}, w_\iota^{(\nu)}, 0)$.

4. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, \\ \quad K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

The rest of the experiment is similar to $\mathsf{hyb}_{0,\nu-1,3,\iota,1}$.

$\mathsf{hyb}_{0,\nu-1,3,\iota,3}$: In this experiment, in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,3,\iota,2}$.
2. Next, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $\underline{j = 1, \ldots, \iota+1}$, it iteratively computes the following:
   - $\mathrm{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1)$
   - $w_j^{(\nu)} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \mathrm{AUX}_j^{(\nu)})$
   - $\mathrm{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
   - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$

   It sets $m_{\iota,0}^{(\nu)} = (v_\iota^{(\nu)}, q_0^{(\nu)}, w_\iota^{(\nu)}, 0)$ and $\underline{m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)}$.
3. It gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, \\ \quad \underline{m_{\iota+1,0}^{(\nu)}}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right),$$

   where the program $\mathsf{Accumulate.Prog}^{(3,\iota,1)}$ is a modification of the program $\mathsf{Accumulate.Prog}^{(3,\iota)}$ (Fig. A.7) and is described in Fig. A.24.

The rest of the experiment if analogous to $\mathsf{Hyb}_{0,\nu-1,3,\iota,2}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota,4}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,3,\iota,3}$ with the only exception that while constructing the ν^{th} constrained key queried by $\mathcal{A}$, $\mathcal{B}$ generates $\underline{(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota,5}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,3,\iota,4}$ with the only exception that $\mathcal{B}$ generates $\underline{\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota,6}$: In this experiment, in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K_{\text{SPS},E}, K_{\text{SPS},F}$, Messages $m_{\iota,0}$, $\underline{m_{\iota+1,0}}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h,i))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h,i))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =\text{'-'}] \wedge [(i > \ell^*) \vee (0 \le i \le \iota) \vee (h \ne h^*)]$, output $\perp$.
Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4.(a) Compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, i+1))$, $(\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
(b) Compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}, (h, i+1))$, $(\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},F})$.
(c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. If $[(h,i) = (h^*, \iota)] \wedge [(m_{\text{IN}} = m_{\iota,0}) \underline{\wedge\ (m_{\text{OUT}} = m_{\iota+1,0})}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
Else if $[(h,i) = (h^*, \iota)] \wedge [(m_{\text{IN}} \ne m_{\iota,0}) \underline{\vee\ (m_{\text{OUT}} \ne m_{\iota+1,0})}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.24. $\mathsf{Accumulate.Prog}^{(3,\iota,1)}$

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator as in $\mathsf{Hyb}_{0,\nu-1,3,\iota,5}$.
2. For $j = 1, \ldots, \iota+1$, it iteratively computes the following:
   - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1)$
   - $w_j^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
   - $\text{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
3. Then, it generates $\underline{(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup\text{-}Enforce}(1^\lambda, n_{\text{ITR}} = 2^\lambda, ((q_0^{(\nu)}, w_0^{(\nu)}, 0), \ldots, (q_0^{(\nu)}, w_\iota^{(\nu)}, 0)))}$.
4. Next, for $j = 1, \ldots, \iota+1$, it iteratively computes $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$.
5. It sets $m_{\iota,0}^{(\nu)} = (v_\iota^{(\nu)}, q_0^{(\nu)}, w_\iota^{(\nu)}, 0)$ and $m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)$.
6. It gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, \\ \qquad m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \qquad K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array} \right).$$

The rest of the experiment if analogous to $\mathsf{Hyb}_{0,\nu-1,3,\iota,5}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota,7}$: In this experiment, to answer the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates everything as in $\mathsf{Hyb}_{0,\nu-1,3,\iota,6}$, however, it hands $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota')}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, \\ \quad K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right),$$

where the program $\mathsf{Accumulate.Prog}^{(3,\iota')}$ is depicted in Fig. A.9. The rest of the experiment is similar to $\mathsf{Hyb}_{0,\nu-1,3,\iota,6}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota,8}$: This experiment is analogous to $\mathsf{hyb}_{0,\nu-1,3,\iota,7}$ with the only exception that while constructing the ν^{th} constrained key queried by $\mathcal{A}$, $\mathcal{B}$ generates $(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$. Notice that this experiment coincides with $\mathsf{Hyb}_{0,\nu-1,3,\iota'}$.

Analysis

Let $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,\vartheta)}(\lambda)$ represents the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in $\mathsf{Hyb}_{0,\nu-1,3,\iota,\vartheta}$, for $\vartheta \in [0, 8]$. From the description of the hybrid experiments it follows that $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,0)}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota')}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,8)}(\lambda)$. Hence, we have

$$|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota')}(\lambda)| \leq \sum_{\vartheta=1}^{8} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,\vartheta-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,\vartheta)}(\lambda)|. \quad \text{(A.7)}$$

Claims A.16–A.23 below will show that the RHS of Eq. (A.7) is negligible and thus Lemma A.8 follows.

Claim A.16. *Assuming* SSB *satisfies the 'index hiding' property defined in Definition 2.3, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,0)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,1)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,0)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,1)}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the the index hiding property of SSB using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\mathrm{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ submits $n_{\text{SSB-BLK}} = 2^\lambda$ and the pair of indices $(i_0^* = 0, i_1^* = \iota)$ to its SSB index hiding challenger $\mathcal{C}$ and receives back a hash key HK, where either $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i_0^* = 0)$ or $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i_1^* = \iota)$.
  2. Next, $\mathcal{B}$ computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.
  3. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  4. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0, 1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. On the other hand, if $b = 1$, then it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$.

5. $\mathcal{B}$ returns the challenge CPRF value y^* to $\mathcal{A}$.

- For $\eta \in [\hat{q}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota,0}$.
- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its SSB index hiding experiment.

Note that if $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i_0^* = 0)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota,0}$. On the other hand, if $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i_1^* = \iota)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota,1}$. This completes the proof of Claim A.16. □

Claim A.17. *Assuming* ACC *is a positional accumulator satisfying the 'indistinguishability of write setup' property defined in Definition 2.4, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,2)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,2)}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the indistinguishability of write setup property of the positional accumulator ACC using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = \iota)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. On the other hand, if $b = 1$, then it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
  4. $\mathcal{B}$ returns the challenge CPRF value y^* to $\mathcal{A}$.
- For $\eta \in [\hat{q}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota,1}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, $\mathcal{B}$ sends $n_{\text{ACC-BLK}} = 2^\lambda$ and the sequence of symbol-index pairs $((x_0^*, 0), \ldots, (x_\iota^*, \iota))$ to its ACC write setup indistinguishability challenger $\mathcal{C}$ and receives back $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0)$, where either $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ or $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enfoce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x_0^*, 0), \ldots, (x_\iota^*, \iota)))$.
  3. Next, it generates $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  4. Then, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0, 0)$. For $j = 1, \ldots, \iota$, it iteratively computes the following:
     - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1)$
     - $w_j = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{j-1}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
     - $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}\text{STORE}_{j-1}, j-1, x_{j-1}^*)$
     - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}, 0))$

     It sets $m_{\iota,0}^{(\nu)} = (v_\iota^{(\nu)}, q_0^{(\nu)}, w_\iota, 0)$.

5. It gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its ACC write setup indistinguishability experiment.

Note that if $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota,1}$. On the other hand, if $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x_0^*, 0), \ldots, (x_\iota^*, \iota)))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota,2}$. This completes the proof of Claim A.17. □

Claim A.18. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, SSB possesses the 'somewhere statistically binding' property defined in Definition 2.3, and ACC is a positional accumulator having the 'write enforcing' property defined in Definition 2.4, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,2)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,3)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The difference between $\mathsf{Hyb}_{0,\nu-1,3,\iota,2}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota,3}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,3,\iota,2}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0)$ within the ν^{th} constrained key provided to $\mathcal{A}$, whereas, in $\mathsf{Hyb}_{0,\nu-1,3,\iota,3}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_1)$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(3,\iota)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.7),
- $P_1 = \mathsf{Accumulate.Prog}^{(3,\iota,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.24).

We will argue that the programs P_0 and P_1 are functionally equivalent, so that, by the security of $\mathcal{IO}$ Claim A.18 follows. The inputs on which the outputs of the two programs can possibly differ are those corresponding to $(h, i) = (h^*, \iota)$. For inputs corresponding to (h^*, ι), the program P_1 performs the additional check '$m_{\mathrm{OUT}} = m_{\iota+1,0}^{(\nu)}$' to determine the type of the outputted signature. We show that this check is redundant by demonstrating that for inputs corresponding to (h^*, ι), if $m_{\mathrm{IN}} = m_{\iota,0}^{(\nu)}$, then either both the programs output $\perp$ or it must hold that $m_{\mathrm{OUT}} = m_{\iota+1,0}^{(\nu)}$ and, therefore, both the programs output signatures of the same type. Notice that $m_{\mathrm{IN}} = m_{\iota,0}^{(\nu)}$ means $v_{\mathrm{IN}} = v_\iota^{(\nu)}, \mathrm{ST} = q_0^{(\nu)}$, and $w_{\mathrm{IN}} = w_\iota^{(\nu)}$. Thus, $v_{\mathrm{OUT}} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_{\mathrm{IN}}, (\mathrm{ST}, w_{\mathrm{IN}}, 0)) = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_\iota^{(\nu)}, (q_0^{(\nu)}, w_\iota^{(\nu)}, 0)) = v_{\iota+1}^{(\nu)}$. Now, recall that in both experiments $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = \iota)$. Therefore, by the somewhere statistically binding property of SSB it follows that $\mathsf{SSB.Verify}(\mathrm{HK}, h^* = \mathcal{H}_{\mathrm{HK}}(x^*), \iota, \mathrm{SYM}_{\mathrm{IN}}, \pi_{\mathrm{SSB}}) = 1$ if and only if $\mathrm{SYM}_{\mathrm{IN}} = x_\iota^*$. Thus, for inputs corresponding to (h^*, ι), both programs will output $\perp$ in case $\mathrm{SYM}_{\mathrm{IN}} \neq x_\iota^*$. Further, in both the experiments, $(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Write}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x_0^*, 0), \ldots, (x_\iota^*, \iota)))$. Therefore, by the write enforcing property of ACC it follows that if $w_{\mathrm{IN}} = w_\iota^{(\nu)}$ and $\mathrm{SYM}_{\mathrm{IN}} = x_\iota^*$, then $w_{\mathrm{OUT}} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}}, \iota, \mathrm{AUX})$ results in $w_{\mathrm{OUT}} = w_{\iota+1}^{(\nu)}$ or $w_{\mathrm{OUT}} = \perp$. In case $w_{\mathrm{OUT}} = \perp$, then clearly both the programs output $\perp$. On the other hand, $w_{\mathrm{OUT}} = w_{\iota+1}^{(\nu)}$ implies $m_{\mathrm{OUT}} = (v_{\mathrm{OUT}}, \mathrm{ST}, w_{\mathrm{OUT}}, 0) = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0) = m_{\iota+1,0}^{(\nu)}$ and the two programs have identical outputs in this case as well. □

Claim A.19. *Assuming* ACC *is a positional accumulator satisfying the 'indistinguishability of write setup' property defined in Definition 2.4, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,4)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim A.19 is similar to that of Claim A.17 with some appropriate modifications which can be readily figured out. □

Claim A.20. *Assuming* SSB *satisfies the 'index hiding' property defined in Definition 2.3, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,4)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,5)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim A.20 is analogous to that of Claim A.16 with certain approximate changes which are easy to determine. □

Claim A.21. *Assuming* ITR *satisfies the 'indistinguishability of enforcing setup' property defined in Definition 2.5, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,5)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,6)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,5)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,6)}(\lambda)|$ is non-negligible. Below, we construct a PPT adversary $\mathcal{B}$ that breaks the indistinguishability of enforcing setup property of the iterator ITR using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. On the other hand, if $b = 1$, then it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
  4. $\mathcal{B}$ returns the challenge CPRF value y^* to $\mathcal{A}$.
- For $\eta \in [\hat{q}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota,5}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it generates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$.
  3. For $j = 1, \ldots, \iota + 1$, it iteratively computes the following:
     - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1)$
     - $w_j^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
     - $\text{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
  4. Then, $\mathcal{B}$ sends $n_{\text{ITR}} = 2^\lambda$ along with the sequence of messages $((q_0^{(\nu)}, w_0^{(\nu)}, 0), \ldots, (q_0^{(\nu)}, w_\iota^{(\nu)}, 0))$ to its ITR enforcing setup indistinguishability challenger $\mathcal{C}$ and receives back $(\text{PP}_{\text{ITR}}, v_0)$, where either $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$ or $(\text{PP}_{\text{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup\text{-}Enforce}(1^\lambda, n_{\text{ITR}} = 2^\lambda, ((q_0^{(\nu)}, w_0^{(\nu)}, 0), \ldots, (q_0^{(\nu)}, w_\iota^{(\nu)}, 0)))$.
  5. For $j = 1, \ldots, \iota + 1$, $\mathcal{B}$ iteratively computes $v_j = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{j-1}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$. It sets $m_{\iota,0}^{(\nu)} = (v_\iota, q_0^{(\nu)}, w_\iota^{(\nu)}, 0)$ and $m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)$.

6. It gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}, v_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, \\ \qquad m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \qquad K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its ITR enforcing setup indistinguishability experiment.

Note that if $(\mathrm{PP}_{\mathrm{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota,5}$. On the other hand, if $(\mathrm{PP}_{\mathrm{ITR}}, v_0) \xleftarrow{\$} \mathsf{ITR.Setup\text{-}Enforce}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda, ((q_0^{(\nu)}, w_0^{(\nu)}, 0), \ldots, (q_0^{(\nu)}, w_\iota^{(\nu)}, 0)))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota,6}$. This completes the proof of Claim A.21. □

Claim A.22. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and ITR has the 'enforcing' property defined in Definition 2.5, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,6)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,7)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The difference between $\mathsf{Hyb}_{0,\nu-1,3,\iota,6}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota,7}$ is the following: In $\mathsf{Hyb}_{0,\nu-1,3,\iota,6}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0)$ within the ν^{th} constrained key provided to $\mathcal{A}$, whereas, in $\mathsf{Hyb}_{0,\nu-1,3,\iota,7}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_1)$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(3,\iota,1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota,0}^{(\nu)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.24),
- $P_1 = \mathsf{Accumulate.Prog}^{(3,\iota')}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}, K_{\mathrm{SPS},F}^{(\nu)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.9).

We will argue that the programs P_0 and P_1 are functionally identical, so that, by the security of $\mathcal{IO}$ Claim A.22 follows. The only inputs on which the outputs of the two programs can possibly differ are those corresponding to $(h, i) = (h^*, \iota)$. For inputs corresponding to (h^*, ι), the program P_0 checks whether '$m_{\mathrm{IN}} = m_{\iota,0}^{(\nu)}$' and '$m_{\mathrm{OUT}} = m_{\iota+1,0}^{(\nu)}$' to determine the type of the outputted signature, while the program P_1 only checks whether '$m_{\mathrm{OUT}} = m_{\iota+1,0}^{(\nu)}$'. Thus, the two programs will be functionally equivalent if we can show that for inputs corresponding to (h^*, ι), $m_{\mathrm{OUT}} = m_{\iota+1,0}^{(\nu)}$ implies $m_{\mathrm{IN}} = m_{\iota,0}^{(\nu)}$. Recall that in both experiment $(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)} \xleftarrow{\$} \mathsf{ITR.Setup\text{-}Enforce}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda, ((q_0^{(\nu)}, w_0^{(\nu)}, 0), \ldots, (q_0^{(\nu)}, w_\iota^{(\nu)}, 0)))$. Now, $m_{\mathrm{OUT}} = m_{\iota+1,0}^{(\nu)}$ implies $v_{\mathrm{OUT}} = v_{\iota+1}^{(\nu)}$. Therefore, by the enforcing property of ITR it follows that $v_{\mathrm{IN}} = v_\iota^{(\nu)}$ and $(\mathrm{ST}, w_{\mathrm{IN}}, 0) = (q_0^{(\nu)}, w_\iota^{(\nu)}, 0)$, which in turn implies that $m_{\mathrm{IN}} = (v_{\mathrm{IN}}, \mathrm{ST}, w_{\mathrm{IN}}, 0) = (v_\iota^{(\nu)}, q_0^{(\nu)}, w_\iota^{(\nu)}, 0) = m_{\iota,0}^{(\nu)}$. □

Claim A.23. *Assuming* ITR *satisfies the 'indistinguishability of enforcing setup' property defined in Definition 2.5, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,7)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota,8)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim A.23 is analogous to that of Claim A.21 with some appropriate modifications which are easy to determine. □

□

Lemma A.9. *Assuming $\mathcal{IO}$ is a a secure indistinguishability obfuscator for* $\mathsf{P/poly}$*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, and* SPS *is a splittable signature scheme satisfying '$\mathrm{VK}_{\mathrm{SPS\text{-}ONE}}$ indistinguishability', '$\mathrm{VK}_{\mathrm{SPS\text{-}ABO}}$ indistinguishability', as well as 'splitting indistinguishability' as defined in Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota')}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota+1)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. In order to establish Lemma A.9, we consider the following sequence of intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,3,\iota'}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota+1}$:

Sequence of Intermediate Hybrids between $\mathsf{Hyb}_{0,\nu-1,3,\iota'}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota+1}$

$\mathsf{Hyb}_{0,\nu-1,3,\iota',0}$: This experiment coincides with $\mathsf{Hyb}_{0,\nu-1,3,\iota'}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',1}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,3,\iota',0}$ except that in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',0}$.
2. Then, it forms the punctured PPRF keys $K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,\iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\mathrm{SPS},E}^{(\nu)},(h^*,\iota+1))$ and $K_{\mathrm{SPS},F}^{(\nu)}\{(h^*,\iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\mathrm{SPS},F}^{(\nu)},(h^*,\iota+1))$.
3. Next, it computes $r_{\mathrm{SPS},G}^{(\nu,\iota+1)} = \mathcal{F}(K_{\mathrm{SPS},E}^{(\nu)},(h^*,\iota+1)), r_{\mathrm{SPS},H}^{(\nu,\iota+1)} = \mathcal{F}(K_{\mathrm{SPS},F}^{(\nu)},(h^*,\iota+1))$, and forms $(\mathrm{SK}_{\mathrm{SPS},G}^{(\nu,\iota+1)},\mathrm{VK}_{\mathrm{SPS},G}^{(\nu,\iota+1)},\mathrm{VK}_{\mathrm{SPS\text{-}REJ},G}^{(\nu,\iota+1)}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\mathrm{SPS},G}^{(\nu,\iota+1)}), (\mathrm{SK}_{\mathrm{SPS},H}^{(\nu,\iota+1)},\mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\iota+1)},\mathrm{VK}_{\mathrm{SPS\text{-}REJ},H}^{(\nu,\iota+1)}) = \mathsf{SPS.Setup1}^\lambda;\ r_{\mathrm{SPS},H}^{(\nu,\iota+1)})$.
4. After that, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $j = 1,\ldots,\iota+1$, it iteratively computes the following:
   - $\mathrm{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1)$
   - $w_j^{(\nu)} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \mathrm{AUX}_j^{(\nu)})$
   - $\mathrm{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
   - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$

   It sets $m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)$.
5. It gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,\iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,\iota+1)\}, \\ \quad K_{\mathrm{SPS},F}^{(\nu)}\{(h^*,\iota+1)\}, \mathrm{SK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, \mathrm{SK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*,\iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*,\iota+1)\}, \\ \quad \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},H}^{(\nu,\iota+1)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right),$$

where the programs $\mathsf{Accumulate.Prog}^{(3,\iota',1)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}$ respectively are the alterations of the programs $\mathsf{Accumulate.Prog}^{(3,\iota')}$ and $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota)}$ (Figs. A.9 and A.8)

and are shown in Figs. A.25 and A.26.

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, Punctured PPRF keys $K_{\text{SPS},E}\{(h^*, \iota+1)\}, K_{\text{SPS},F}\{(h^*, \iota+1)\}$, Signing keys SK_G, SK_H, Verification keys VK_G, VK_H, Message $m_{\iota+1,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*, \iota+1)\}, (h,i)), (\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.

(b) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \iota+1)\}, (h,i)), (\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
Else, set $\text{VK}_{\text{SPS},F} = \text{VK}_H$.

(c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.

(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.

(e) If $[\alpha =\text{'-'}] \ \wedge\ [(i > \ell^*) \ \vee\ (0 \leq i \leq \iota) \ \vee\ (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =\text{'-'}] \ \wedge\ [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.

(f) If $\alpha =$'-', output $\perp$.

2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.

3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.

(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.

4.(a) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*, \iota+1)\}, (h, i+1)), (\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
Else, set $\text{SK}'_{\text{SPS},E} = \text{SK}_G$.

(b) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \iota+1)\}, (h, i+1)), (\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},F})$.
Else, set $\text{SK}'_{\text{SPS},F} = \text{SK}_H$.

(c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$.
If $[(h,i) = (h^*, \iota)] \ \wedge\ [m_{\text{OUT}} = m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
Else if $[(h,i) = (h^*, \iota)] \ \wedge\ [m_{\text{OUT}} \neq m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.

5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.25. $\mathsf{Accumulate.Prog}^{(3,\iota',1)}$

$\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$: This experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,3,\iota',1}$ with the only exception that while constructing the ν^{th} constrained key queried by $\mathcal{A}$, $\mathcal{B}$ selects $r^{(\nu,\iota+1)}_{\text{SPS},G}, r^{(\nu,\iota+1)}_{\text{SPS},H} \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, i.e., in other words, $\mathcal{B}$ generates $(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},G}), (\text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},H}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',3}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$ except that in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$.
2. Then, it creates the punctured PPRF keys $K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},E}, (h^*, \iota+1))$ and $K^{(\nu)}_{\text{SPS},F}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},F}, (h^*, \iota+1))$.

Constants: PPRF keys $K_{\text{SPS},A}$, $K_{\text{SPS},B}$, Punctured PPRF keys $K_{\text{SPS},E}\{(h^*, \iota+1)\}$, $K_{\text{SPS},F}\{(h^*, \iota+1)\}$, Verification keys VK_G, VK_H, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) If $(h, \ell_{\text{INP}}) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*, \iota+1)\}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.

(b) If $(h, \ell_{\text{INP}}) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \iota+1)\}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
Else, set $\text{VK}_{\text{SPS},F} = \text{VK}_H$.

(c) Set $m = (v, \text{ST}, w, 0)$ and α='-'.

(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set α ='E'.

(e) If $[\alpha$ ='-'$] \ \wedge\ [(\ell_{\text{INP}} > \ell^*) \ \vee\ (0 < \ell_{\text{INP}} \leq \iota) \ \vee\ (h \neq h^*)]$, output $\perp$.
Else if $[\alpha$ ='-'$] \ \wedge\ [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m, \sigma_{\text{SPS,IN}}) = 1]$, set α ='F'.

(f) If α ='-', output $\perp$.

2.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.

(b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},B})$.

(c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \ \wedge\ [\alpha$ ='F'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.26. $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}$

3. After that, it generates $(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},G}), (\text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},H}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$.
4. Then, it computes $m^{(\nu)}_{\iota+1,0} = (v^{(\nu)}_{\iota+1}, q^{(\nu)}_0, w^{(\nu)}_{\iota+1}, 0)$ just as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$.
5. Next, it forms $(\sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m^{(\nu)}_{\iota+1,0},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-ONE},G}, \text{SK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, m^{(\nu)}_{\iota+1,0})$ and $(\sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m^{(\nu)}_{\iota+1,0},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-ONE},H}, \text{SK}^{(\nu,\iota+1)}_{\text{SPS-ABO},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-ABO},H}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, m^{(\nu)}_{\iota+1,0})$.
6. It gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\}, \\ \quad K^{(\nu)}_{\text{SPS},F}\{(h^*, \iota+1)\}, \sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m^{(\nu)}_{\iota+1,0},G}, \text{SK}^{(\nu,\iota+1)}_{\text{SPS-ABO},H}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, m^{(\nu)}_{\iota+1,0}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}\{(h^*, \iota+1)\}, K^{(\nu)}_{\text{SPS},F}\{(h^*, \iota+1)\}, \\ \quad \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},H}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, \\ \quad K^{(\nu)}_{\text{SPS},B}, h^*, \ell^*]) \end{array}\right),$$

where the program $\mathsf{Accumulate.Prog}^{(3,\iota',2)}$ is an alteration of the program $\mathsf{Accumulate.Prog}^{(3,\iota',1)}$ (Fig. A.25) and is shown in Fig. A.27.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',4}$: In this experiment, in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates everything as in $\mathsf{hyb}_{0,\nu-1,3,\iota',3}$,

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, Punctured PPRF keys $K_{\text{SPS},E}\{(h^*, \iota{+}1)\}, K_{\text{SPS},F}\{(h^*, \iota{+}1)\}$, Signature σ_G, Signing key SK_H, Verification keys VK_G, VK_H, Message $m_{\iota+1,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,\iota+1)\}, (h,i)), (\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.
(b) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*,\iota+1)\}, (h,i)), (\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
Else, set $\text{VK}_{\text{SPS},F} = \text{VK}_H$.
(c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =\text{'-'}] \wedge [(i > \ell^*) \vee (0 \leq i \leq \iota) \vee (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4.(a) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,\iota+1)\}, (h,i+1)), (\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
(b) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*,\iota+1)\}, (h,i+1)), (\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},F})$.
Else, set $\text{SK}'_{\text{SPS},F} = \text{SK}_H$.
(c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. If $[(h,i) = (h^*, \iota)] \wedge [m_{\text{OUT}} = m_{\iota+1,0}]$, set $\sigma_{\text{SPS,OUT}} = \sigma_G$.
Else if $[(h,i) = (h^*, \iota)] \wedge [m_{\text{OUT}} \neq m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.27. $\mathsf{Accumulate.Prog}^{(3,\iota',2)}$

however, it hands $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*,\iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*,\iota+1)\}, \\ \quad K_{\text{SPS},F}^{(\nu)}\{(h^*,\iota+1)\}, \sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}, \text{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \underline{\text{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}}, \text{VK}_{\text{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*,\iota+1)\}, K_{\text{SPS},F}^{(\nu)}\{(h^*,\iota+1)\}, \\ \quad \underline{\text{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}}, \text{VK}_{\text{SPS},H}^{(\nu,\iota+1)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, \\ \quad K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,3,\iota',3}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',5}$: In this experiment, in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ forms all the components just as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',4}$,

however, it gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \underline{\mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}}, m_{\iota+1,0}^{(\nu)}, \\ \quad h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \underline{\mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\mathrm{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, \\ \quad K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,3,\iota',4}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',6}$: In this experiment, to answer the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ creates all the components as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',5}$, however, it returns the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',3)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, \overline{K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}}, \\ \quad \overline{K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\mathrm{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, \\ \quad K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right)$$

to $\mathcal{A}$, where program $\mathsf{Accumulate.Prog}^{(3,\iota',3)}$ is a modification of program $\mathsf{Accumulate.Prog}^{(3,\iota',2)}$ (Fig. A.27) and is depicted in Fig. A.28. The remaining part of the experiment is identical to $\mathsf{hyb}_{0,\nu-1,3,\iota',5}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',7}$: In this experiment, to answer the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the components exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',6}$ except that it does not generate $(\sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},H}, \mathrm{VK}_{\text{SPS-ONE},H}^{(\nu,\iota+1)}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}) \xleftarrow{\$}$

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, Punctured PPRF keys $K_{\text{SPS},E}\{(h^*, \iota{+}1)\}, K_{\text{SPS},F}\{(h^*, \iota{+}1)\}$, Signature σ_G, Signing key SK_H, Verification keys VK_G, VK_H, Message $m_{\iota+1,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol SYM_{IN}, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\text{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,\iota+1)\}, (h,i)), (\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.
(b) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*,\iota+1)\}, (h,i)), (\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
Else, set $\text{VK}_{\text{SPS},F} = \text{VK}_H$.
(c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}, w_{\text{IN}}, 0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =\text{'-'}] \wedge [(i > \ell^*) \vee (0 \leq i \leq \iota) \vee (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =\text{'-'}] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\text{HK}, h, i, \text{SYM}_{\text{IN}}, \pi_{\text{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, i, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}, w_{\text{IN}}, 0))$.
4.(a) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*,\iota+1)\}, (h,i+1)), (\text{SK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS},E}, \text{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},E})$.
(b) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*,\iota+1)\}, (h,i+1)), (\text{SK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS},F}, \text{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},F})$.
Else, set $\text{SK}'_{\text{SPS},F} = \text{SK}_H$.
(c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}, w_{\text{OUT}}, 0)$. If $[(h,i) = (h^*, \iota)] \wedge [m_{\text{OUT}} = m_{\iota+1,0}]$, set $\sigma_{\text{SPS,OUT}} = \sigma_G$.
Else if $[(h,i) = (h^*, \iota)] \wedge [m_{\text{OUT}} \neq m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
Else if $[(h,i) = (h^*, \iota+1)] \wedge [m_{\text{IN}} = m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
Else if $[(h,i) = (h^*, \iota+1)] \wedge [m_{\text{IN}} \neq m_{\iota+1,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},F}, m_{\text{OUT}})$.
Else if $i < \ell^*$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},E}, m_{\text{OUT}})$.
5. Output $(w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}})$.

Fig. A.28. $\mathsf{Accumulate.Prog}^{(3,\iota',3)}$

$\mathsf{SPS.Split}(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},H}, m^{(\nu)}_{\iota+1,0})$ and provides $\mathcal{A}$ with the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}\{(h^*,\iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',3)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}\{(h^*,\iota+1)\}, \\ \quad K^{(\nu)}_{\text{SPS},F}\{(h^*,\iota+1)\}, \sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m^{(\nu)}_{\iota+1,0},G}, \underline{\text{SK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-ONE},G}, \underline{\text{VK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}}, m^{(\nu)}_{\iota+1,0}, \\ \quad h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K^{(\nu)}_{\text{SPS},A}, K^{(\nu)}_{\text{SPS},B}, K^{(\nu)}_{\text{SPS},E}\{(h^*,\iota+1)\}, K^{(\nu)}_{\text{SPS},F}\{(h^*,\iota+1)\} \\ \quad \text{VK}^{(\nu,\iota+1)}_{\text{SPS-ONE},G}, \underline{\text{VK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\text{SPS},A}, \\ \quad {}^{(\nu)}_{\text{SPS},B}, h^*, \ell^*]) \end{array}\right).$$

The rest of the experiment is the same as $\mathsf{Hyb}_{0,\nu-1,3,\iota',6}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',8}$: In this experiment, in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the components as in $\mathsf{hyb}_{0,\nu-1,3,\iota',7}$,

however, it returns the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',4)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad \underline{K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \mathrm{SK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad \underline{\mathrm{VK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, h^*, \ell^*}]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\mathrm{SPS},A}^{(\nu)}, \\ \quad K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right)$$

to $\mathcal{A}$, where the programs $\mathsf{Accumulate.Prog}^{(3,\iota',4)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,2)}$ respectively are the modifications of the programs $\mathsf{Accumulate.Prog}^{(3,\iota',3)}$ and $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}$ (Figs. A.28 and A.26) and are shown in Figs. A.29 and A.30. The rest of the experiment if identical to $\mathsf{Hyb}_{0,\nu-1,3,\iota',7}$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',9}$: This experiments analogous to $\mathsf{hyb}_{0,\nu-1,3,\iota',8}$ with the only exception that while

Constants: Maximum number of blocks for SSB hash $n_{\text{SSB-BLK}} = 2^\lambda$, SSB hash key HK, Public parameters for positional accumulator $\mathrm{PP}_{\mathrm{ACC}}$, Public parameters for iterator $\mathrm{PP}_{\mathrm{ITR}}$, Punctured PPRF keys $K_{\mathrm{SPS},E}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}\{(h^*, \iota+1)\}$, Signing key SK_G, Verification key VK_G, Message $m_{\iota+1,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Index i, Symbol $\mathrm{SYM}_{\mathrm{IN}}$, TM state ST, Accumulator value w_{IN}, Auxiliary value AUX, Iterator value v_{IN}, Signature $\sigma_{\mathrm{SPS,IN}}$, SSB hash value h, SSB opening value π_{SSB}

Output: (Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS-OUT}}$), or $\perp$

1.(a) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\mathrm{SPS},E} = \mathcal{F}(K_{\mathrm{SPS},E}\{(h^*, \iota+1)\}, (h,i)), (\mathrm{SK}_{\mathrm{SPS},E}, \mathrm{VK}_{\mathrm{SPS},E}, \mathrm{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r_{\mathrm{SPS},E})$.
Else, set $\mathrm{VK}_{\mathrm{SPS},E} = \mathrm{VK}_G$.
(b) If $(h,i) \neq (h^*, \iota+1)$, compute $r_{\mathrm{SPS},F} = \mathcal{F}(K_{\mathrm{SPS},F}\{(h^*, \iota+1)\}, (h,i)), (\mathrm{SK}_{\mathrm{SPS},F}, \mathrm{VK}_{\mathrm{SPS},F}, \mathrm{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r_{\mathrm{SPS},F})$.
(c) Set $m_{\mathrm{IN}} = (v_{\mathrm{IN}}, \mathrm{ST}, w_{\mathrm{IN}}, 0)$ and $\alpha =$'-'.
(d) If $\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},E}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1$, set $\alpha =$'E'.
(e) If $[\alpha =$'-'$] \wedge [(i > \ell^*) \vee \underline{(0 \leq i \leq \iota+1)} \vee (h \neq h^*)]$, output $\perp$.
Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\mathrm{VK}_{\mathrm{SPS},F}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1]$, set $\alpha =$'F'.
(f) If $\alpha =$'-', output $\perp$.
2. If $\mathsf{SSB.Verify}(\mathrm{HK}, h, i, \mathrm{SYM}_{\mathrm{IN}}, \pi_{\mathrm{SSB}}) = 0$, output $\perp$.
3.(a) Compute $w_{\mathrm{OUT}} = \mathsf{ACC.Update}(\mathrm{PP}_{\mathrm{ACC}}, w_{\mathrm{IN}}, \mathrm{SYM}_{\mathrm{IN}}, i, \mathrm{AUX})$. If $w_{\mathrm{OUT}} = \perp$, output $\perp$.
(b) Compute $v_{\mathrm{OUT}} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\mathrm{ITR}}, v_{\mathrm{IN}}, (\mathrm{ST}, w_{\mathrm{IN}}, 0))$.
4.(a) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\mathrm{SPS},E} = \mathcal{F}(K_{\mathrm{SPS},E}\{(h^*, \iota+1)\}, (h, i+1)), (\mathrm{SK}'_{\mathrm{SPS},E}, \mathrm{VK}'_{\mathrm{SPS},E}, \mathrm{VK}'_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\mathrm{SPS},E})$.
(b) If $(h,i) \neq (h^*, \iota)$, compute $r'_{\mathrm{SPS},F} = \mathcal{F}(K_{\mathrm{SPS},F}\{(h^*, \iota+1)\}, (h, i+1)), (\mathrm{SK}'_{\mathrm{SPS},F}, \mathrm{VK}'_{\mathrm{SPS},F}, \mathrm{VK}'_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\mathrm{SPS},F})$.
(c) Set $m_{\mathrm{OUT}} = (v_{\mathrm{OUT}}, \mathrm{ST}, w_{\mathrm{OUT}}, 0)$. If $\underline{(h,i) = (h^*, \iota)}$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\underline{\mathrm{SK}_G, m_{\mathrm{OUT}}})$.
Else if $[(h,i) = (h^*, \iota+1)] \wedge [m_{\mathrm{IN}} = m_{\iota+1,0}]$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},E}, m_{\mathrm{OUT}})$.
Else if $[(h,i) = (h^*, \iota+1)] \wedge [m_{\mathrm{IN}} \neq m_{\iota+1,0}]$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},F}, m_{\mathrm{OUT}})$.
Else if $i < \ell^*$, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},\alpha}, m_{\mathrm{OUT}})$.
Else, compute $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},E}, m_{\mathrm{OUT}})$.
5. Output $(w_{\mathrm{OUT}}, v_{\mathrm{OUT}}, \sigma_{\mathrm{SPS,OUT}})$.

Fig. A.29. $\mathsf{Accumulate.Prog}^{(3,\iota',4)}$

Constants: PPRF keys $K_{\text{SPS},A}$, $K_{\text{SPS},B}$, Punctured PPRF keys $K_{\text{SPS},E}\{(h^*, \iota+1)\}$, $K_{\text{SPS},F}\{(h^*, \iota+1)\}$, Verification key VK_G, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) If $(h, \ell_{\text{INP}}) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}\{(h^*, \iota+1)\}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
Else, set $\text{VK}_{\text{SPS},E} = \text{VK}_G$.
(b) If $(h, \ell_{\text{INP}}) \neq (h^*, \iota+1)$, compute $r_{\text{SPS},F} = \mathcal{F}(K_{\text{SPS},F}\{(h^*, \iota+1)\}, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SPS},F}, \text{VK}_{\text{SPS},F}, \text{VK}_{\text{SPS-REJ},F}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},F})$.
(c) Set $m = (v, \text{ST}, w, 0)$ and α='-'.
(d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 1$, set α ='E'.
(e) If $[\alpha$ ='-'$]\ \wedge\ [(\ell_{\text{INP}} > \ell^*)\ \vee\ (0 < \ell_{\text{INP}} \leq \iota+1)\ \vee\ (h \neq h^*)]$, output $\perp$.
Else if $[\alpha$ ='-'$]\ \wedge\ [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},F}, m, \sigma_{\text{SPS,IN}}) = 1]$, set α ='F'.
(f) If α ='-', output $\perp$.
2.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
(b) Compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}, (h, \ell_{\text{INP}}, 0))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{DSPS},B})$.
(c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)]\ \wedge\ [\alpha$ ='F'$]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.30. $\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,2)}$

constructing the ν^{th} constrained key queried by $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates $(\text{SK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS},G}, \text{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},G}) = \mathsf{SPS.Setup}(1^\lambda;\ r^{(\nu,\iota+1)}_{\text{SPS},G} = \mathcal{F}(K^{(\nu)}_{\text{SPS},E}, (h^*, \iota+1)))$.

$\mathsf{Hyb}_{0,\nu-1,3,\iota',10}$: This experiment corresponds to $\mathsf{hyb}_{0,\nu-1,3,\iota+1}$.

Analysis

Let $\mathsf{Adv}^{(0,\nu-1,3,\iota',\vartheta)}_{\mathcal{A}}(\lambda)$ represents the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in $\mathsf{Hyb}_{0,\nu-1,3,\iota',\vartheta}$, for $\vartheta \in [0, 10]$. From the description of the hybrid experiments it follows that $\mathsf{Adv}^{(0,\nu-1,3,\iota')}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,\nu-1,3,\iota',0)}_{\mathcal{A}}(\lambda)$ and $\mathsf{Adv}^{(0,\nu-1,3,\iota+1)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,\nu-1,3,\iota',10)}_{\mathcal{A}}(\lambda)$. Hence, we have

$$|\mathsf{Adv}^{(0,\nu-1,3,\iota')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota+1)}_{\mathcal{A}}(\lambda)| \leq \sum_{\vartheta=1}^{10} |\mathsf{Adv}^{(0,\nu-1,3,\iota',\vartheta-1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota',\vartheta)}_{\mathcal{A}}(\lambda)|. \quad \text{(A.8)}$$

Claims A.24–A.33 below will show that the RHS of Eq. (A.8) is negligible and thus Lemma A.9 follows.

Claim A.24. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$ satisfies the correctness under puncturing property defined in Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,3,\iota',0)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota',1)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The only difference between $\mathsf{Hyb}_{0,\nu-1,3,\iota',0}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota',1}$ is the following:
In $\mathsf{Hyb}_{0,\nu-1,3,\iota',0}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_0)$ and $\mathcal{IO}(P'_0)$ within the ν^{th} constrained key provided to $\mathcal{A}$, while in $\mathsf{hyb}_{0,\nu-1,3,\iota',1}$, it includes the programs $\mathcal{IO}(P_1)$ and $\mathcal{IO}(P'_1)$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(3,\iota')}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}, K^{(\nu)}_{\text{SPS},F}, m^{(\nu)}_{\iota+1,0}, h^*, \ell^*]$ (Fig. A.9),

- $P_0' = \mathsf{Change\text{-}SPSProg}^{(3,\iota)}[K^{(\nu)}_{\mathrm{SPS},A}, K^{(\nu)}_{\mathrm{SPS},B}, K^{(\nu)}_{\mathrm{SPS},E}, K^{(\nu)}_{\mathrm{SPS},F}, h^*, \ell^*]$ (Fig. A.8),
- $P_1 = \mathsf{Accumulate.Prog}^{(3,\iota',1)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K^{(\nu)}_{\mathrm{SPS},E}\{(h^*, \iota+1)\}, K^{(\nu)}_{\mathrm{SPS},F}\{(h^*, \iota+1)\}, \mathrm{SK}^{(\nu,\iota+1)}_{\mathrm{SPS},G}, \mathrm{SK}^{(\nu,\iota+1)}_{\mathrm{SPS},H}, \mathrm{VK}^{(\nu,\iota+1)}_{\mathrm{SPS},G}, \mathrm{VK}^{(\nu,\iota+1)}_{\mathrm{SPS},H}, m^{(\nu)}_{\iota+1,0}, h^*, \ell^*]$ (Fig. A.25),
- $P_1' = \mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K^{(\nu)}_{\mathrm{SPS},A}, K^{(\nu)}_{\mathrm{SPS},B}, K^{(\nu)}_{\mathrm{SPS},E}\{(h^*, \iota+1)\}, K^{(\nu)}_{\mathrm{SPS},F}\{(h^*, \iota+1)\}, \mathrm{VK}^{(\nu,\iota+1)}_{\mathrm{SPS},G}, \mathrm{VK}^{(\nu,\iota+1)}_{\mathrm{SPS},H}, h^*, \ell^*]$ (Fig. A.26).

Observe that by the correctness under puncturing property of the PPRF $\mathcal{F}$, the programs P_0 and P_1 are functionally identical for all inputs corresponding to $(h, i) \neq (h^*, \iota)$ and $(h, i) \neq (h^*, \iota + 1)$. For inputs corresponding to (h^*, ι),the program P_1 uses the hardwired signing keys which are exactly same as those computed by the program P_0. The same is true for the hardwired verification keys used by P_1 for inputs corresponding to $(h^*, \iota + 1)$. Thus, the programs P_0 and P_1 are functionally equivalent. A similar argument shows that the same is correct for programs P_0' and P_1'. Therefore, by the security of $\mathcal{IO}$ Claim A.24 follows. Ofcourse, we need to consider a sequence of intermediate hybrid experiments to switch the programs one at a time. □

Claim A.25. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,3,\iota',1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota',2)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}^{(0,\nu-1,3,\iota',1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,3,\iota',2)}_{\mathcal{A}}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ is given below. We note that in the following we work in a model of selective pseudorandomness for PPRF involving two independent punctured keys and two challenge values for a challenge input, one under each key. However, this model is clearly equivalent to the original single punctured key and single challenge value model described in Definition 2.2 through a hybrid argument.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x^*_0 \ldots x^*_{\ell^*-1} \in \mathcal{X}_{\mathrm{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0, 1\}$. If $b = 0$, then $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. On the other hand, if $b = 1$, then it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$.
  4. $\mathcal{B}$ returns the challenge CPRF value y^* to $\mathcal{A}$.
- For $\eta \in [\hat{q}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',1}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\mathrm{SPS},A}, K^{(\nu)}_{\mathrm{SPS},B} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it creates $(\mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, w^{(\nu)}_0, \mathrm{STORE}^{(\nu)}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, v^{(\nu)}_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$.
  3. $\mathcal{B}$ sends $(h^*, \iota+1)$ as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back two punctured PPRF keys $K^*_1\{(h^*, \iota + 1)\}, K^*_2\{(h^*, \iota + 1)\}$ and two values $r^*_1, r^*_2 \in \mathcal{Y}_{\mathrm{PPRF}}$, where either $r^*_1 = \mathcal{F}(K^*_1, (h^*, \iota + 1)), r^*_2 = \mathcal{F}(K^*_2, (h^*, \iota + 1))$ or $r^*_1, r^*_2 \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$. $\mathcal{B}$ *implicitly* views the keys K^*_1 and K^*_2 as the keys $K^{(\nu)}_{\mathrm{SPS},E}$ and $K^{(\nu)}_{\mathrm{SPS},F}$ respectively.
  4. $\mathcal{B}$ generates $(\mathrm{SK}^{(\nu,\iota+1)}_{\mathrm{SPS},G}, \mathrm{VK}^{(\nu,\iota+1)}_{\mathrm{SPS},G}, \mathrm{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},G}) = \mathsf{SPS.Setup}(1^\lambda;\ r^*_1)$ and $(\mathrm{SK}^{(\nu,\iota+1)}_{\mathrm{SPS},H}, \mathrm{VK}^{(\nu,\iota+1)}_{\mathrm{SPS},H}, \mathrm{VK}^{(\nu,\iota+1)}_{\text{SPS-REJ},H}) = \mathsf{SPS.Setup}(1^\lambda;\ r^*_2)$.
  5. Next, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$. For $j = 1, \ldots, \iota + 1$, it iteratively computes the following:

- $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1)$
- $w_j^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
- $\text{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
- $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$

It sets $m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)$.

6. It gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_1^*\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_1^*\{(h^*, \iota+1)\}, \\ \qquad K_2^*\{(h^*, \iota+1)\}, \text{SK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{SK}_{\text{SPS},H}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_1^*\{(h^*, \iota+1)\}, K_2^*\{(h^*, \iota+1)\}, \\ \qquad \text{VK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS},H}^{(\nu,\iota+1)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \qquad K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its PPRF selective pseudorandomness experiment.

Note that if $r_1^* = \mathcal{F}(K_1^*, (h^*, \iota+1)), r_2^* = \mathcal{F}(K_2^*, (h^*, \iota+1))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',1}$. On the other hand, if $r_1^*, r_2^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, the $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$. This completes the proof of Claim A.25. □

Claim A.26. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',2)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',3)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The only difference between $\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota',3}$ is the following:
In $\mathsf{Hyb}_{0,\nu-1,3,\iota',2}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0)$ within the ν^{th} constrained key provided to $\mathcal{A}$, while in $\mathsf{hyb}_{0,\nu-1,3,\iota',3}$, it includes the program $\mathcal{IO}(P_1)$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(3,\iota',1)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \text{SK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{SK}_{\text{SPS},H}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.25),
- $P_1 = \mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\nu)}, \text{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}^{(\nu,\iota+1)}, \text{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.27).

Now, the only inputs on which the outputs of the two programs can possibly differ are those corresponding to $(h, i) = (h^*, \iota)$. However, observe that for inputs corresponding to (h^*, ι), if $m_{\text{OUT}} = m_{\iota+1,0}^{(\nu)}$, then both programs clearly output the same signature, where P_0 computes the signature explicitly and P_1 has the signature hardwired into it. On the other hand, by the correctness [Property (ii)] of the splittable signature SPS defined in Definition 2.6 it follows that the programs P_0 and P_1 output same signatures even when $m_{\text{OUT}} \neq m_{\iota+1,0}^{(\nu)}$ for inputs corresponding to (h^*, ι). Hence, the two programs are functionally equivalent. Therefore, Claim A.26 follows by the security of $\mathcal{IO}$. □

Claim A.27. *Assuming* SPS *is a splittable signature scheme satisfying '$\text{VK}_{\text{SPS-ONE}}$ indistinguishability' as per Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',4)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',4)}(\lambda)|$ is non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the $\mathrm{VK}_{\text{SPS-ONE}}$ indistinguishability of SPS using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. On the other hand, if $b = 1$, then it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
  4. $\mathcal{B}$ returns the challenge CPRF value y^* to $\mathcal{A}$.
- For $\eta \in [\hat{q}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',3}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it creates $(\mathrm{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\mathrm{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. Then, $\mathcal{B}$ creates the punctured PPRF keys $K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},E}^{(\nu)}, (h^*, \iota+1))$ and $K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},F}^{(\nu)}, (h^*, \iota+1))$.
  4. Next, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $j = 1, \ldots, \iota+1$, it iteratively computes the following:
     - $\mathrm{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\mathrm{PP}_{\text{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1)$
     - $w_j^{(\nu)} = \mathsf{ACC.Update}(\mathrm{PP}_{\text{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \mathrm{AUX}_j^{(\nu)})$
     - $\mathrm{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\mathrm{PP}_{\text{ACC}}^{(\nu)}, \mathrm{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
     - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\mathrm{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$

     It sets $m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)$.
  5. After that, $\mathcal{B}$ sends $m_{\iota+1,0}^{(\nu)}$ as the challenge message to its SPS $\mathrm{VK}_{\text{SPS-ONE}}$ indistinguishability challenger $\mathcal{C}$ and receives back a signature-verification key pair $(\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK})$, where VK is either a normal verification key VK_{SPS} or a one verification key $\mathrm{VK}_{\text{SPS-ONE}}$ for the message $m_{\iota+1,0}^{(\nu)}$.
  6. $\mathcal{B}$ generates $(\mathrm{SK}_{\text{SPS},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-REJ},H}^{(\nu,\iota+1)}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$ and forms $(\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ONE},H}^{(\nu,\iota+1)}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}) \xleftarrow{\$} \mathsf{SPS.Split}(\mathrm{SK}_{\text{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)})$.
  7. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$
\mathrm{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l}
\mathrm{HK}, \mathrm{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}, \\
\mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\
\mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\text{ACC}}^{(\nu)}, \mathrm{PP}_{\text{ITR}}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, \\
\quad K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \mathrm{VK}, \mathrm{VK}_{\text{SPS},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\
\mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \\
\quad \mathrm{VK}, \mathrm{VK}_{\text{SPS},H}^{(\nu,\iota+1)}, h^*, \ell^*]), \\
\mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\text{ACC}}^{(\nu)}, \mathrm{PP}_{\text{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\
\quad K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, h^*, \ell^*])
\end{array} \right).
$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its SPS $\text{VK}_{\text{SPS-ONE}}$ indistinguishability experiment.

Notice that if $\text{VK} = \text{VK}_{\text{SPS}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',3}$. On the other hand, if $\text{VK} = \text{VK}_{\text{SPS-ONE}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',4}$. This completes the proof of Claim A.27. □

Claim A.28. *Assuming* SPS *is a splittable signature scheme satisfying* '$\text{VK}_{\text{SPS-ABO}}$ *indistinguishability' as per Definition 2.6, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',4)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',5)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',4)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',5)}(\lambda)|$ is non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the $\text{VK}_{\text{SPS-ABO}}$ indistinguishability of SPS using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. On the other hand, if $b = 1$, then it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
  4. $\mathcal{B}$ returns the challenge CPRF value y^* to $\mathcal{A}$.
- For $\eta \in [\hat{q}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',4}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it creates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. Then, $\mathcal{B}$ creates the punctured PPRF keys $K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},E}^{(\nu)}, (h^*, \iota+1))$ and $K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},F}^{(\nu)}, (h^*, \iota+1))$.
  4. Next, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $j = 1, \ldots, \iota+1$, it iteratively computes the following:
     - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1)$
     - $w_j^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
     - $\text{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
     - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$

     It sets $m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)$.
  5. After that, $\mathcal{B}$ sends $m_{\iota+1,0}^{(\nu)}$ as the challenge message to its SPS $\text{VK}_{\text{SPS-ABO}}$ indistinguishability challenger $\mathcal{C}$ and receives back an all-but-one signing key-verification key pair $(\text{SK}_{\text{SPS-ABO}}, \text{VK})$, where VK is either a normal verification key VK_{SPS} or an all-but-one verification key $\text{VK}_{\text{SPS-ABO}}$.
  6. $\mathcal{B}$ generates $(\text{SK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS-REJ},G}^{(\nu,\iota+1)}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$ and forms $(\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \text{SK}_{\text{SPS-ABO},G}^{(\nu,\iota+1)}, \text{VK}_{\text{SPS-ABO},G}^{(\nu,\iota+1)}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}_{\text{SPS},G}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)})$.

7. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}, \mathrm{SK}_{\text{SPS-ABO}}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \mathrm{VK}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \mathrm{VK}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its SPS $\mathrm{VK}_{\text{SPS-ABO}}$ indistinguishability experiment.

Notice that if $\mathrm{VK} = \mathrm{VK}_{\mathrm{SPS}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',4}$. On the other hand, if $\mathrm{VK} = \mathrm{VK}_{\text{SPS-ABO}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',5}$. This completes the proof of Claim A.28. □

Claim A.29. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for $\mathsf{P/poly}$, for any PPT adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',5)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',6)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function negl.*

Proof. The only difference between $\mathsf{Hyb}_{0,\nu-1,3,\iota',5}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota',6}$ is the following:
In $\mathsf{Hyb}_{0,\nu-1,3,\iota',5}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0)$ within the ν^{th} constrained key returned to $\mathcal{A}$, while in $\mathsf{hyb}_{0,\nu-1,3,\iota',6}$, it includes the program $\mathcal{IO}(P_1)$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(3,\iota',2)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.27),
- $P_1 = \mathsf{Accumulate.Prog}^{(3,\iota',3)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}, \mathrm{SK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ONE},G}^{(\nu,\iota+1)}, \mathrm{VK}_{\text{SPS-ABO},H}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.28).

We will argue that the programs P_0 and P_1 are functionally equivalent, so that, by the security of $\mathcal{IO}$ Claim A.29 holds. First of all observe that the constants hardwired in both the programs are identically generated. Clearly, the inputs on which the outputs of the programs P_0 and P_1 can possibly differ are those corresponding to $(h, i) = (h^*, \iota+1)$. For inputs corresponding to $(h^*, \iota+1)$, let us consider the following two cases:

(I) $(m_{\mathrm{IN}} = m_{\iota+1,0}^{(\nu)})$: In this case, using the correctness [Properties (i), (iii) and (vi)] of the splitable signature SPS described in Definition 2.6 it follows that for both programs either $\alpha =$'-' or $\alpha =$'E'. Now, if $\alpha =$'-', then both programs output $\perp$. On the other hand, if $\alpha =$'E', then P_0 outputs the signature $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},\alpha}, m_{\mathrm{OUT}}) = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},E}, m_{\mathrm{OUT}})$, which is the same signature that P_1 is programmed to output in this case. Thus, both programs have identical outputs in this case.

(II) $(m_{\mathrm{IN}} \neq m_{\iota+1,0}^{(\nu)})$: In this case, we use the correctness [Property (v)] of SPS described in Definition 2.6 to conclude that $\alpha \neq$'E' and correctness [Properties (i) and (iv)] of SPS confirms that either $\alpha =$'-' or $\alpha =$'F'. Now, if α='-', then both programs output $\perp$ as earlier. Otherwise, if $\alpha =$'F', then P_0 outputs $\sigma_{\mathrm{SPS,OUT}} = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},\alpha}, m_{\mathrm{OUT}}) = \mathsf{SPS.Sign}(\mathrm{SK}'_{\mathrm{SPS},F}, m_{\mathrm{OUT}})$, which P_1 is programmed to output in this case. Therefore, both programs are functionally equivalent in this case as well.

□

Claim A.30. *Assuming* SPS *is a splitable signature scheme satisfying 'splitting indistinguishability' as per Definition 2.6, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',6)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',7)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',6)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',7)}(\lambda)|$ is non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the splitting indistinguishability of SPS using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. On the other hand, if $b = 1$, then it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
  4. $\mathcal{B}$ returns the challenge CPRF value y^* to $\mathcal{A}$.
- For $\eta \in [\hat{q}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,3,\iota',6}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)}, K_{\text{SPS},F}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, it creates $(\text{PP}_{\text{ACC}}^{(\nu)}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. Then, $\mathcal{B}$ generates the punctured PPRF keys $K_{\text{SPS},E}^{(\nu)}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},E}^{(\nu)}, (h^*, \iota+1))$ and $K_{\text{SPS},F}^{(\nu)}\{(h^*, \iota+1)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},F}^{(\nu)}, (h^*, \iota+1))$.
  4. Next, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0^{(\nu)}, 0)$. For $j = 1, \ldots, \iota+1$, it iteratively computes the following:
     - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1)$
     - $w_j^{(\nu)} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}^{(\nu)}, w_{j-1}^{(\nu)}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
     - $\text{STORE}_j^{(\nu)} = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}^{(\nu)}, \text{STORE}_{j-1}^{(\nu)}, j-1, x_{j-1}^*)$
     - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}^{(\nu)}, 0))$

     It sets $m_{\iota+1,0}^{(\nu)} = (v_{\iota+1}^{(\nu)}, q_0^{(\nu)}, w_{\iota+1}^{(\nu)}, 0)$.
  5. After that, $\mathcal{B}$ sends $m_{\iota+1,0}^{(\nu)}$ as the challenge message to its SPS splitting indistinguishability challenger $\mathcal{C}$ and receives back a tuple $(\sigma^*_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \text{VK}^*_{\text{SPS-ONE}}, \text{SK}^*_{\text{SPS-ABO}}, \text{VK}^*_{\text{SPS-ABO}})$, where
     - either $(\sigma^*_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \text{VK}^*_{\text{SPS-ONE}}, \text{SK}^*_{\text{SPS-ABO}}, \text{VK}^*_{\text{SPS-ABO}}) = (\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}})$
     - or $(\sigma^*_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \text{VK}^*_{\text{SPS-ONE}}, \text{SK}^*_{\text{SPS-ABO}}, \text{VK}^*_{\text{SPS-ABO}}) = (\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \text{VK}_{\text{SPS-ONE}}, \text{SK}'_{\text{SPS-ABO}}, \text{VK}'_{\text{SPS-ABO}})$

     such that $(\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \text{VK}_{\text{SPS-ONE}}, \text{SK}_{\text{SPS-ABO}}, \text{VK}_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}_{\text{SPS}}, m_{\iota+1,0}^{(\nu)})$,
     $(\sigma'_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \text{VK}'_{\text{SPS-ONE}}, \text{SK}'_{\text{SPS-ABO}}, \text{VK}'_{\text{SPS-ABO}}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}'_{\text{SPS}}, m_{\iota+1,0}^{(\nu)})$,
     SK_{SPS} and SK'_{SPS} being two independently generated signing keys for SPS.

6. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}^{(3,\iota',3)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma^*_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{SK}^*_{\text{SPS-ABO}}, \mathrm{VK}^*_{\text{SPS-ONE},}, \mathrm{VK}^*_{\text{SPS-ABO}}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \\ \quad \mathrm{VK}^*_{\text{SPS-ONE}}, \mathrm{VK}^*_{\text{SPS-ABO}}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\mathrm{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its SPS splitting indistinguishability experiment.

Notice that if $(\sigma^*_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}^*_{\text{SPS-ONE}}, \mathrm{SK}^*_{\text{SPS-ABO}}, \mathrm{VK}^*_{\text{SPS-ABO}}) = (\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}_{\text{SPS-ONE}}, \mathrm{SK}'_{\text{SPS-ABO}}, \mathrm{VK}'_{\text{SPS-ABO}})$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',6}$. On the other hand, if $(\sigma^*_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}^*_{\text{SPS-ONE}}, \mathrm{SK}^*_{\text{SPS-ABO}}, \mathrm{VK}^*_{\text{SPS-ABO}}) = (\sigma_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)}}, \mathrm{VK}_{\text{SPS-ONE}}, \mathrm{SK}_{\text{SPS-ABO}}, \mathrm{VK}_{\text{SPS-ABO}})$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,3,\iota',7}$. This completes the proof of Claim A.30. □

Claim A.31. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',7)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',8)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The only difference between $\mathsf{Hyb}_{0,\nu-1,3,\iota',7}$ and $\mathsf{Hyb}_{0,\nu-1,3,\iota',8}$ is the following:
In $\mathsf{Hyb}_{0,\nu-1,3,\iota',7}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_0)$ and $\mathcal{IO}(P_0')$ within the ν^{th} constrained key returned to $\mathcal{A}$, while in $\mathsf{hyb}_{0,\nu-1,3,\iota',8}$, it includes the programs $\mathcal{IO}(P_1)$ and $\mathcal{IO}(P_1')$ instead, where

- $P_0 = \mathsf{Accumulate.Prog}^{(3,\iota',3)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}, \mathrm{SK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}, \mathrm{VK}^{(\nu,\iota+1)}_{\text{SPS-ONE},G}, \mathrm{VK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.28),
- $P_0' = \mathsf{Change\text{-}SPS.Prog}^{(3,\iota,1)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \mathrm{VK}^{(\nu,\iota=1)}_{\text{SPS-ONE},G}, \mathrm{VK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}, h^*, \ell^*]$ (Fig. A.26),
- $P_1 = \mathsf{Accumulate.Prog}^{(3,\iota',4)}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \mathrm{SK}^{(\nu,\iota+1)}_{\mathrm{SPS},G}, \mathrm{VK}^{(\nu,\iota+1)}_{\mathrm{SPS},G}, m_{\iota+1,0}^{(\nu)}, h^*, \ell^*]$ (Fig. A.29),
- $P_1' = \mathsf{Change\text{-}SPS.Prog}^{(3,\iota,2)}[K_{\mathrm{SPS},A}^{(\nu)}, K_{\mathrm{SPS},B}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}\{(h^*, \iota+1)\}, K_{\mathrm{SPS},F}^{(\nu)}\{(h^*, \iota+1)\}, \mathrm{VK}^{(\nu,\iota=1)}_{\mathrm{SPS},G}, h^*, \ell^*]$ (Fig. A.30).

We will argue that the programs P_0 and P_1, as well as , the programs P_0' and P_1' are functionally equivalent, so that, by the security of $\mathcal{IO}$ Claim A.31 follows. First consider the programs P_0 and P_1. Clearly the only inputs on which the outputs of the two programs can possibly differ are those corresponding to $(h,i) = (h^*, \iota)$ and $(h,i) = (h^*, \iota+1)$. Now, for inputs corresponding to (h^*, ι), the outputs of the two programs are identical due to the correctness [Property (ii)] of the splitable signature SPS described in Definition 2.6 and the fact that the hardwired signature $\sigma^{(\nu,\iota+1)}_{\text{SPS-ONE},m_{\iota+1,0}^{(\nu)},G}$ and the all-but-one signing key $\mathrm{SK}^{(\nu,\iota+1)}_{\text{SPS-ABO},G}$ used by the program P_0 for inputs corresponding to (h^*, ι) are obtained by running $\mathsf{SPS.Split}(\mathrm{SK}^{(\nu,\iota+1)}_{\mathrm{SPS},G}, m_{\iota+1,0}^{(\nu)})$, while the hardwired signing key used by P_1 in this case is $\mathrm{SK}^{(\nu,\iota+1)}_{\mathrm{SPS},G}$. Similarly, for inputs corresponding to $(h^*, \iota+1)$,

the outputs of the two programs are also identical because of the correctness [Properties (i), (iii), (v), (iv) and (vi)] of SPS and the fact that the hardwired one and all-but-one verification keys used by the program P_0 for inputs corresponding to $(h^*, \iota+1)$ are generated by running $\mathsf{SPS.Split}(\mathrm{SK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}, m_{\iota+1,0}^{(\nu)})$, while the hardwired verification key used by the program P_1 in this case is $\mathrm{VK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}$, which is the matching verification key of $\mathrm{SK}_{\mathrm{SPS},G}^{(\nu,\iota+1)}$. Hence, the two programs are functionally equivalent. The same type of argument holds for the programs P_0' and P_1'. □

Claim A.32. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',8)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',9)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim A.32 is analogous to that of Claim A.25 with some appropriate modifications that are readily identifiable. □

Claim A.33. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$ satisfies the correctness under puncturing property defined in Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',9)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,\iota',10)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim A.33 is similar to that of Claim A.24 with some appropriate modifications which are easy to find out. □

□

Lemma A.10. *Assuming $\mathcal{IO}$ is a a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, and* SPS *is a secure splitable signature scheme satisfying 'VK$_{\text{SPS-ONE}}$ indistinguishability', 'VK$_{\text{SPS-ABO}}$ indistinguishability', as well as 'splitting indistinguishability' as defined in Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,(\ell^*-1)')}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Lemma A.10 proceeds along the same line as that of Lemma A.9 with certain appropriate changes which can be readily determined. □

Lemma A.11. *Assuming $\mathcal{IO}$ is a a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* ACC *is a secure positional accumulator possessing the 'indistinguishability of read setup' as well as 'read enforcing' properties defined in Definition 2.4, and* SPS *is a secure splitable signature scheme satisfying 'VK$_{\text{SPS-ONE}}$ indistinguishability', 'VK$_{\text{SPS-ABO}}$ indistinguishability', as well as 'splitting indistinguishability' as defined in Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4,0')}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. In order to establish Lemma A.11, we consider the following sequence of intermediate hybrid experiments between $\mathsf{Hyb}_{0,\nu-1,4}$ and $\mathsf{Hyb}_{0,\nu-1,4,0'}$:

Sequence of Intermediate Hybrids between $\mathsf{Hyb}_{0,\nu-1,4}$ and $\mathsf{Hyb}_{0,\nu-1,4,0'}$

$\mathsf{Hyb}_{0,\nu-1,4\text{-I}}$: This experiment coincides with $\mathsf{Hyb}_{0,\nu-1,4}$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-II}}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,4\text{-I}}$ except that in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,4\text{-I}}$.

2. Then, it creates the punctured PPRF keys $K^{(\nu)}_{\text{SPS},A}\{(h^*,\ell^*,0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},A},(h^*,\ell^*,0))$ and $K^{(\nu)}_{\text{SPS},B}\{(h^*,\ell^*,0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},B},(h^*,\ell^*,0))$.
3. After that it computes $r^{(\nu,0)}_{\text{SPS},C} = \mathcal{F}(K^{(\nu)}_{\text{SPS},A},(h^*,\ell^*,0))$, $r^{(\nu,0)}_{\text{SPS},D} = \mathcal{F}(K^{(\nu)}_{\text{SPS},B},(h^*,\ell^*,0))$, and forms $(\text{SK}^{(\nu,0)}_{\text{SPS},C},\text{VK}^{(\nu,0)}_{\text{SPS},C},\text{VK}^{(\nu,0)}_{\text{SPS-REJ},C}) = \mathsf{SPS.Setup}(1^\lambda;\ r^{(\nu,0)}_{\text{SPS},C})$, $(\text{SK}^{(\nu,0)}_{\text{SPS},D},\text{VK}^{(\nu,0)}_{\text{SPS},D},\text{VK}^{(\nu,0)}_{\text{SPS-REJ},D}) = \mathsf{SPS.Setup}(1^\lambda;\ r^{(\nu,0)}_{\text{SPS},D})$.
4. Next, it sets $m^{(\nu)}_{0,0} = (v^{(\nu)}_0, q^{(\nu)}_0, w^{(\nu)}_0, 0)$. For $j = 1,\ldots,\ell^*$, it iteratively computes the following:
   - $\text{AUX}^{(\nu)}_j = \mathsf{ACC.Prep\text{-}Write}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1)$
   - $w^{(\nu)}_j = \mathsf{ACC.Update}(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_{j-1}, x^*_{j-1}, j-1, \text{AUX}^{(\nu)}_j)$
   - $\text{STORE}^{(\nu)}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}^{(\nu)}_{\text{ACC}}, \text{STORE}^{(\nu)}_{j-1}, j-1, x^*_{j-1})$
   - $v^{(\nu)}_j = \mathsf{ITR.Iterate}(\text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_{j-1}, (q^{(\nu)}_0, w^{(\nu)}_{j-1}, 0))$

   It sets $m^{(\nu)}_{\ell^*,0} = (v^{(\nu)}_{\ell^*}, q^{(\nu)}_0, w^{(\nu)}_{\ell^*}, 0)$.
5. It gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,1)}[K^{(\nu)}_{\text{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\text{SPS},B}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\text{SPS},E}, \text{SK}^{(\nu,0)}_{\text{SPS},C}, \\ \quad \text{SK}^{(\nu,0)}_{\text{SPS},D}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1,0,1)}_{\text{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, \\ \quad K^{(\nu)}_{\text{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\text{SPS},B}\{(h^*,\ell^*,0)\}, \text{VK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS},D}, h^*, \ell^*]) \end{array}\right),$$

where the programs $\mathsf{Change\text{-}SPS.Prog}^{(4,1)}$ and $\mathsf{Constrained\text{-}Key.Prog}^{(1,0,1)}_{\text{CPRF}}$ respectively are the modifications of the programs $\mathsf{Change\text{-}SPS.Prog}^{(4)}$ and $\mathsf{Constrained\text{-}Key.Prog}^{(1)}_{\text{CPRF}}$ (Figs. A.10 and A.2) and are depicted in Figs. A.31 and A.32.

Constants: Punctured PPRF keys $K_{\text{SPS},A}\{(h^*,\ell^*,0)\}, K_{\text{SPS},B}\{(h^*,\ell^*,0)\}$, PPRF key $K_{\text{SPS},E}$, Signing keys SK_C, SK_D, Message $m_{\ell^*,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E},(h,\ell_{\text{INP}}),(\text{SK}_{\text{SPS},E},\text{VK}_{\text{SPS},E},\text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Set $m = (v, \text{ST}, w, 0)$.
(c) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
2.(a) If $(h,\ell_{\text{INP}}) \neq (h^*,\ell^*)$, compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*,\ell^*,0)\},(h,\ell_{\text{INP}},0)),(\text{SK}_{\text{SPS},A},\text{VK}_{\text{SPS},A},\text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
Else, set $\text{SK}_{\text{SPS},A} = \text{SK}_C$.
(b) If $(h,\ell_{\text{INP}}) \neq (h^*,\ell^*)$, compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*,\ell^*,0)\},(h,\ell_{\text{INP}},0)),(\text{SK}_{\text{SPS},B},\text{VK}_{\text{SPS},B},\text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},B})$.
Else, set $\text{SK}_{\text{SPS},B} = \text{SK}_D$.
(c) If $[(h,\ell_{\text{INP}}) = (h^*,\ell^*)] \wedge [m \neq m_{\ell^*,0}]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.31. $\mathsf{Change\text{-}SPS.Prog}^{(4,1)}$

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda$, Punctured PPRF keys $K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}$, Verification keys VK_C, VK_D, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) If $(h, \ell_{\text{INP}}, t) \neq (h^*, \ell^*, 1)$, compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   Else, set $\text{VK}_{\text{SPS},A} = \text{VK}_C$.
   (b) If $(h, \ell_{\text{INP}}, t) \neq (h^*, \ell^*, 1)$, compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},B})$.
   Else, set $\text{VK}_{\text{SPS},B} = \text{VK}_D$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \ \wedge \ [(t > t^*) \ \vee \ (h \neq h^*) \ \vee \ (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \ \wedge \ [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \ \wedge \ [\alpha =$'$B$'$]$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, output $\mathcal{F}(K, (h, \ell_{\text{INP}}))$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.32. $\mathsf{Constrained\text{-}Key.Prog}^{(1,0,1)}_{\text{CPRF}}$

$\mathsf{Hyb}_{0,\nu-1,4\text{-III}}$: This experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-II}}$ with the only exception that while constructing the ν^{th} constrained key queried by $\mathcal{A}$, $\mathcal{B}$ selects $r^{(\nu,0)}_{\text{SPS},C}, r^{(\nu,0)}_{\text{SPS},D} \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, i.e., in other words, $\mathcal{B}$ generates $(\text{SK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},C})$, $(\text{SK}^{(\nu,0)}_{\text{SPS},D}, \text{VK}^{(\nu,0)}_{\text{SPS},D}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},D}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-IV}}$: This experiment is similar to $\mathsf{Hyb}_{0,\nu-1,4\text{-III}}$ except that in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:

1. It first generates all the PPRF keys as well as the public parameters for the positional accumulator and the iterator as in $\mathsf{Hyb}_{0,\nu-1,4\text{-III}}$.
2. Then, it creates the punctured PPRF keys $K^{(\nu)}_{\text{SPS},A}\{(h^*, \ell^*, 0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},A}, (h^*, \ell^*, 0))$ and $K^{(\nu)}_{\text{SPS},B}\{(h^*, \ell^*, 0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K^{(\nu)}_{\text{SPS},B}, (h^*, \ell^*, 0))$.
3. Next, it forms $(\text{SK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},C})$, $(\text{SK}^{(\nu,0)}_{\text{SPS},D}, \text{VK}^{(\nu,0)}_{\text{SPS},D}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},D}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$.
4. After that, it computes $m^{(\nu)}_{\ell^*,0} = (v^{(\nu)}_{\ell^*}, q^{(\nu)}_0, w^{(\nu)}_{\ell^*}, 0)$ just as in $\mathsf{Hyb}_{0,\nu-1,4\text{-III}}$.

5. Next, it generates $(\sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{\ell^*,0},C}, \text{VK}^{(\nu,0)}_{\text{SPS-ONE},C}, \text{SK}^{(\nu,0)}_{\text{SPS-ABO},C}, \text{VK}^{(\nu,0)}_{\text{SPS-ABO},C}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}^{(\nu,0)}_{\text{SPS},C}, m^{(\nu)}_{\ell^*,0})$ and $(\sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{\ell^*,0},D}, \text{VK}^{(\nu,0)}_{\text{SPS-ONE},D}, \text{SK}^{(\nu,0)}_{\text{SPS-ABO},D}, \text{VK}^{(\nu,0)}_{\text{SPS-ABO},D}) \xleftarrow{\$} \mathsf{SPS.Split}(\text{SK}^{(\nu,0)}_{\text{SPS},D}, m^{(\nu)}_{\ell^*,0})$.
6. It gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K^{(\nu)}_{\text{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\text{SPS},B}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\text{SPS},E}, \sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{\ell^*,0},C}, \\ \quad \underline{\text{SK}^{(\nu,0)}_{\text{SPS-ABO},D}}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1,0,1)}_{\text{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, \\ \quad K^{(\nu)}_{\text{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\text{SPS},B}\{(h^*,\ell^*,0)\}, \text{VK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS},D}, h^*, \ell^*]) \end{array}\right),$$

where the program $\mathsf{Change\text{-}SPS.Prog}^{(4,2)}$ is an alteration of the program $\mathsf{Change\text{-}SPS.Prog}^{(4,1)}$ (Fig. A.31) and is shown in Fig. A.33.

Constants: Punctured PPRF keys $K_{\text{SPS},A}\{(h^*,\ell^*,0)\}, K_{\text{SPS},B}\{(h^*,\ell^*,0)\}$, PPRF key $K_{\text{SPS},E}$, Signature $\underline{\sigma_C}$, Signing key SK_D, Message $m_{\ell^*,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}), (\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Set $m = (v, \text{ST}, w, 0)$.
(c) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
2.(a) If $(h, \ell_{\text{INP}}) \neq (h^*, \ell^*)$, compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*,\ell^*,0)\}, (h, \ell_{\text{INP}}, 0)), (\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
(b) If $(h, \ell_{\text{INP}}) \neq (h^*, \ell^*)$, compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*,\ell^*,0)\}, (h, \ell_{\text{INP}}, 0)), (\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},B})$.
Else, set $\text{SK}_{\text{SPS},B} = \text{SK}_D$.
(c) If $[(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [m \neq m_{\ell^*,0}]$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},B}, m)$.
$\underline{\text{Else if } [(h, \ell_{\text{INP}} = (h^*, \ell^*)] \wedge [m = m_{\ell^*,0}], \text{ output } \sigma_{\text{SPS,OUT}} = \sigma_C.}$
$\underline{\text{Else, output } \sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m).}$

Fig. A.33. $\mathsf{Change\text{-}SPS.Prog}^{(4,2)}$

$\mathbf{Hyb_{0,\nu-1,4\text{-}V}}$: In this experiment, in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the components as in $\mathsf{hyb}_{0,\nu-1,4\text{-IV}}$,

however, it hands $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K_{\mathrm{SPS},A}^{(\nu)}\{(h^*, \ell^*, 0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*, \ell^*, 0)\}, K_{\mathrm{SPS},E}^{(\nu)}, \sigma^{(\nu,0)}_{\text{SPS-ONE},m_{\ell^*,0}^{(\nu)},C}, \\ \quad \mathrm{SK}_{\text{SPS-ABO},D}^{(\nu,0)}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1,0,1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}\{(h^*, \ell^*, 0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*, \ell^*, 0)\}, \underline{\mathrm{VK}_{\text{SPS-ONE},C}^{(\nu,0)}}, \mathrm{VK}_{\mathrm{SPS},D}^{(\nu,0)}, h^*, \ell^*]) \end{array}\right),$$

The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-IV}}$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-VI}}$: In this experiment, in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ creates all the components as in $\mathsf{hyb}_{0,\nu-1,4\text{-V}}$, however, it hands $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K_{\mathrm{SPS},A}^{(\nu)}\{(h^*, \ell^*, 0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*, \ell^*, 0)\}, K_{\mathrm{SPS},E}^{(\nu)}, \sigma^{(\nu,0)}_{\text{SPS-ONE},m_{\ell^*,0}^{(\nu)},C}, \\ \quad \mathrm{SK}_{\text{SPS-ABO},D}^{(\nu,0)}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1,0,1)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}\{(h^*, \ell^*, 0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*, \ell^*, 0)\}, \mathrm{VK}_{\text{SPS-ONE},C}^{(\nu,0)}, \underline{\mathrm{VK}_{\text{SPS-ABO},D}^{(\nu,0)}}, h^*, \ell^*]) \end{array}\right),$$

The rest of the experiment is similar to $\mathsf{Hyb}_{0,\nu-1,4\text{-V}}$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-VII}}$: In this experiment, to answer the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the components just as in $\mathsf{Hyb}_{0,\nu-1,4\text{-VI}}$, but it provides $\mathcal{A}$ with the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K_{\mathrm{SPS},A}^{(\nu)}\{(h^*, \ell^*, 0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*, \ell^*, 0)\}, K_{\mathrm{SPS},E}^{(\nu)}, \sigma^{(\nu,0)}_{\text{SPS-ONE},m_{\ell^*,0}^{(\nu)},C}, \\ \quad \mathrm{SK}_{\text{SPS-ABO},D}^{(\nu,0)}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1,0,2)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)},} \\ \quad \underline{K_{\mathrm{SPS},A}^{(\nu)}\{(h^*, \ell^*, 0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*, \ell^*, 0)\}, \mathrm{VK}_{\text{SPS-ONE},C}^{(\nu,0)}, \mathrm{VK}_{\text{SPS-ABO},D}^{(\nu,0)}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]}) \end{array}\right),$$

where program $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1,0,2)}$ is a modification of program $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1,0,1)}$ (Fig. A.32) and is shown in Fig. A.34. The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-VI}}$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-VIII}}$: In This experiment is the same as $\mathsf{Hyb}_{0,\nu-1,4\text{-VII}}$ with the only exception that

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda$, Punctured PPRF keys $K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}$, Verification keys VK_C, VK_D, Message $m_{\ell^*,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) If $(h, \ell_{\text{INP}}, t) \neq (h^*, \ell^*, 1)$, compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   Else, set $\text{VK}_{\text{SPS},A} = \text{VK}_C$.
   (b) If $(h, \ell_{\text{INP}}, t) \neq (h^*, \ell^*, 1)$, compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   Else, set $\text{VK}_{\text{SPS},B} = \text{VK}_D$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \ \wedge \ [(t > t^*) \ \vee \ (h \neq h^*) \ \vee \ (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \ \wedge \ [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \ \wedge \ [\alpha =$'$B$'$]$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, output $\mathcal{F}(K, (h, \ell_{\text{INP}}))$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   If $[(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, 1)] \ \wedge \ [m_{\text{IN}} = m_{\ell^*,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
   Else if $[(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, 1)] \ \wedge \ [m_{\text{IN}} \neq m_{\ell^*,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},B}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.34. $\mathsf{Constrained\text{-}Key.Prog}^{(1,0,2)}_{\text{CPRF}}$

while creating the ν^{th} constrained key queried by $\mathcal{A}$, $\mathcal{B}$ generates $(\text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x^*_0, 0), \ldots, (x^*_{\ell^*-1}, \ell^* - 1)), i^* = 0)$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-IX}}$: In this experiment, to answer the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ generates all the components just as in $\mathsf{hyb}_{0,\nu-1,4\text{-VIII}}$,

however, it gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K_{\mathrm{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},E}^{(\nu)}, \sigma^{(\nu,0)}_{\text{SPS-ONE},m_{\ell^*,0}^{(\nu)},C}, \\ \quad \mathrm{SK}_{\text{SPS-ABO},D}^{(\nu,0)}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1,0,3)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, \mathrm{VK}_{\text{SPS-ONE},C}^{(\nu,0)}, \mathrm{VK}_{\text{SPS-ABO},D}^{(\nu,0)}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]}) \end{array}\right),$$

where program $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1,0,3)}$ is a modification of program $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1,0,2)}$ (Fig. A.35) and is shown in Fig. A.35. The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-VIII}}$.

$\mathsf{hyb}_{0,\nu-1,4\text{-X}}$: This experiment is identical to $\mathsf{Hyb}_{0,\nu-1,4\text{-IX}}$ with the only exception that while constructing the ν^{th} constrained key queried by $\mathcal{A}$, $\mathcal{B}$ forms $\underline{(\mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)}$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-XI}}$: In this experiment, in response to the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ creates all the components as in $\mathsf{Hyb}_{0,\nu-1,4\text{-X}}$ except that it does not generate $(\sigma^{(\nu,0)}_{\text{SPS-ONE},\ m_{\ell^*,0}^{(\nu)},D},\ \mathrm{VK}_{\text{SPS-ONE},D}^{(\nu,0)},\ \mathrm{SK}_{\text{SPS-ABO},D}^{\nu,0)},\ \mathrm{VK}_{\text{SPS-ABO},D}^{(\nu,0)}) \xleftarrow{\$} \underline{\mathsf{SPS.Split}(\mathrm{SK}_{\text{SPS},D}^{(\nu,0)},\ m_{\ell^*,0}^{(\nu)})}$ and hands $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, w_0^{(\nu)}, \mathrm{STORE}_0^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, v_0^{(\nu)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\nu)}, w_0^{(\nu)}, v_0^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K_{\mathrm{SPS},E}^{(\nu)}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K_{\mathrm{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},E}^{(\nu)}, \sigma^{(\nu,0)}_{\text{SPS-ONE},m_{\ell^*,0}^{(\nu)},C}, \\ \quad \underline{\mathrm{SK}_{\text{SPS-ABO},C}^{(\nu,0)}}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{CPRF}}^{(1,0,3)}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}^{(\nu)}, \mathrm{PP}_{\mathrm{ITR}}^{(\nu)}, K, K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, \\ \quad K_{\mathrm{SPS},A}^{(\nu)}\{(h^*,\ell^*,0)\}, K_{\mathrm{SPS},B}^{(\nu)}\{(h^*,\ell^*,0)\}, \mathrm{VK}_{\text{SPS-ONE},C}^{(\nu,0)}, \underline{\mathrm{VK}_{\text{SPS-ABO},C}^{(\nu,0)}}, m_{\ell^*,0}^{(\nu)}, h^*, \ell^*]) \end{array}\right).$$

The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-X}}$.

$\mathsf{hyb}_{0,\nu-1,4\text{-XII}}$: In this experiment, to answer the ν^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\nu)} \in \mathbb{M}_\lambda$ with $M^{(\nu)}(x^*) = 0$, $\mathcal{B}$ creates all the components as in $\mathsf{Hyb}_{0,\nu-1,4\text{-XI}}$, but it

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda$, Punctured PPRF keys $K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}$, Verification keys VK_C, VK_D, Message $m_{\ell^*,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) If $(h, \ell_{\text{INP}}, t) \neq (h^*, \ell^*, 1)$, compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   Else, set $\text{VK}_{\text{SPS},A} = \text{VK}_C$.
   (b) If $(h, \ell_{\text{INP}}, t) \neq (h^*, \ell^*, 1)$, compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   Else, set $\text{VK}_{\text{SPS},B} = \text{VK}_D$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \wedge [(t > t^*) \vee (h \neq h^*) \vee (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =$'$B$'$]$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =$'$A$'$] \wedge [(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [t \leq 1]$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, output $\mathcal{F}(K, (h, \ell_{\text{INP}}))$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   If $[(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, 1)] \wedge [m_{\text{IN}} = m_{\ell^*,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
   Else if $[(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, 1)] \wedge [m_{\text{IN}} \neq m_{\ell^*,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},B}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.35. $\mathsf{Constrained\text{-}Key.Prog}^{(1,0,3)}_{\text{CPRF}}$

provides $\mathcal{A}$ with the constrained key

$$\text{SK}_{\text{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, w^{(\nu)}_0, \text{STORE}^{(\nu)}_0, \text{PP}^{(\nu)}_{\text{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w^{(\nu)}_0, v^{(\nu)}_0, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K^{(\nu)}_{\text{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,3)}[K^{(\nu)}_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, K^{(\nu)}_{\text{SPS},E}, \text{SK}^{(\nu,0)}_{\text{SPS},C}, h^*, \ell^*]), \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}^{(1,0,4)}_{\text{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \text{PP}^{(\nu)}_{\text{ACC}}, \text{PP}^{(\nu)}_{\text{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda,} \\ \quad \underline{K^{(\nu)}_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, K^{(\nu)}_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, \text{VK}^{(\nu,0)}_{\text{SPS},C}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*])} \end{array} \right),$$

where the programs $\mathsf{Change\text{-}SPS.Prog}^{(4,3)}$ and $\mathsf{Constrained\text{-}Key.Prog}^{(1,0,4)}_{\text{CPRF}}$ are the alterations of the programs $\mathsf{Change\text{-}SPS.Prog}^{(4,2)}$ and $\mathsf{Constrained\text{-}Key.Prog}^{(1,0,3)}_{\text{CPRF}}$ (Figs. A.33 and A.35) and

are shown in Figs. A.36 and A.37 respectively. The rest of the experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-XI}}$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-XIII}}$: This experiment is analogous to $\mathsf{Hyb}_{0,\nu-1,4\text{-XII}}$ except that while creating the

Constants: Punctured PPRF key $K_{\text{SPS},A}\{(h^*,\ell^*,0)\}$, PPRF key $K_{\text{SPS},E}$, Signing key SK_C, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: TM state ST, Accumulator value w, Iterator value v, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Signature $\sigma_{\text{SPS,OUT}}$, or $\perp$

1.(a) Compute $r_{\text{SPS},E} = \mathcal{F}(K_{\text{SPS},E}, (h, \ell_{\text{INP}}), (\text{SK}_{\text{SPS},E}, \text{VK}_{\text{SPS},E}, \text{VK}_{\text{SPS-REJ},E}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},E})$.
(b) Set $m = (v, \text{ST}, w, 0)$.
(c) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},E}, m, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
2.(a) If $(h, \ell_{\text{INP}}) \neq (h^*, \ell^*)$, compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*,\ell^*,0)\}, (h, \ell_{\text{INP}}, 0)), (\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
(b) If $(h, \ell_{\text{INP}}) = (h^*, \ell^*)$, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_C, m)$.
Else, output $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}_{\text{SPS},A}, m)$.

Fig. A.36. $\mathsf{Change\text{-}SPS.Prog}^{(4,3)}$

ν^{th} constrained key queried by $\mathcal{A}$, $\mathcal{B}$ and forms $(\text{SK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS},C}, \text{VK}^{(\nu,0)}_{\text{SPS-REJ},C}) = \mathsf{SPS.Setup}(1^\lambda; r^{(\nu,0)}_{\text{SPS},C} = \mathcal{F}(K^{(\nu)}_{\text{SPS},A}, (h^*, \ell^*, 0)))$.

$\mathsf{Hyb}_{0,\nu-1,4\text{-XIV}}$: This experiment corresponds to $\mathsf{Hyb}_{0,\nu-1,4,0'}$.

Analysis

Let $\mathsf{Adv}^{(0,\nu-1,4\text{-}\vartheta)}_{\mathcal{A}}(\lambda)$ represents the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in $\mathsf{Hyb}_{0,\nu-1,4\text{-}\vartheta}$, for $\vartheta \in \{\text{I}, \ldots, \text{XIV}\}$. From the description of the hybrid experiments it follows that $\mathsf{Adv}^{(0,\nu-1,4)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,\nu-1,4\text{-I})}_{\mathcal{A}}(\lambda)$ and $\mathsf{Adv}^{(0,\nu-1,4,0')}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,\nu-1,4\text{-XIV})}_{\mathcal{A}}(\lambda)$. Hence, we have

$$|\mathsf{Adv}^{(0,\nu-1,4)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4,0')}_{\mathcal{A}}(\lambda)| \leq \sum_{\vartheta=\text{II}}^{\text{XIV}} |\mathsf{Adv}^{(0,\nu-1,4\text{-}(\vartheta-\text{I}))}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4\text{-}\vartheta)}_{\mathcal{A}}(\lambda)|. \quad \text{(A.9)}$$

Claims A.34–A.46 below will show that the RHS of Eq. (A.9) is negligible and thus Lemma A.11 follows.

Claim A.34. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$ satisfies the correctness under puncturing property defined in Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,4\text{-}I)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4\text{-}II)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Claim A.34 uses a similar kind of logic as that employed in the proof of Claim A.24. We omit the details here. □

Claim A.35. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,4\text{-}II)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4\text{-}III)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Running time on challenge input t^*, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda$, Punctured PPRF keys $K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}$, Verification key VK_C, Message $m_{\ell^*,0}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: CPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, or Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \wedge [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) If $(h, \ell_{\text{INP}}, t) \neq (h^*, \ell^*, 1)$, compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},A})$.
   Else, set $\text{VK}_{\text{SPS},A} = \text{VK}_C$.
   (b) If $(h, \ell_{\text{INP}}, t) \neq (h^*, \ell^*, 1)$, compute $r_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},B}, \text{VK}_{\text{SPS},B}, \text{VK}_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r_{\text{SPS},B})$.
   (c) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$ and $\alpha =$'-'.
   (d) If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1$, set $\alpha =$'A'.
   (e) If $[\alpha =$'-'$] \wedge [(t > t^*) \vee (t \leq 1) \vee (h \neq h^*) \vee (\ell_{\text{INP}} \neq \ell^*)]$, output $\perp$.
   Else if $[\alpha =$'-'$] \wedge [\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},B}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 1]$, set $\alpha =$'B'.
   (f) If $\alpha =$'-', output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =$'$B$'$]$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \wedge [\alpha =$'$A$'$] \wedge [(h, \ell_{\text{INP}}) = (h^*, \ell^*)] \wedge [t \leq 1]$, output $\perp$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, output $\mathcal{F}(K, (h, \ell_{\text{INP}}))$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},A})$.
   (b) Compute $r'_{\text{SPS},B} = \mathcal{F}(K_{\text{SPS},B}\{(h^*, \ell^*, 0)\}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS},B}, \text{VK}'_{\text{SPS-REJ},B}) = \mathsf{SPS.Setup}(1^\lambda; r'_{\text{SPS},B})$.
   (c) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   If $[(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, 1)] \wedge [m_{\text{IN}} = m_{\ell^*,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
   Else if $[(h, \ell_{\text{INP}}, t) = (h^*, \ell^*, 1)] \wedge [m_{\text{IN}} \neq m_{\ell^*,0}]$, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},B}, m_{\text{OUT}})$.
   Else, compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},\alpha}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. A.37. $\mathsf{Constrained\text{-}Key.Prog}^{(1,0,4)}_{\text{CPRF}}$

Proof. The proof of Claim A.35 resembles that of Claim A.25 with some suitable changes. The details are omitted. □

Claim A.36. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,4\text{-}III)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4\text{-}IV)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. Claim A.36 can be proven using an analogous logic as that used in the proof of Claim A.26. We omit the details here. □

Claim A.37. *Assuming* SPS *is a splittable signature scheme satisfying '$\text{VK}_{\text{SPS-ONE}}$ indistinguishability' as per Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1,4\text{-}IV)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4\text{-}V)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of Claim A.37 is similar to that of Claim A.27 and, therefore, we do not provide the details here. □

Claim A.38. *Assuming* SPS *is a splittable signature scheme satisfying* '$\text{VK}_{\text{SPS-ABO}}$ *indistinguishability' as per Definition 2.6, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}V)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}VI)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim A.38 proceeds along a similar path to that of Claim A.28. We omit the details here. □

Claim A.39. *Assuming* $\mathcal{IO}$ *is a secure indistinguishability obfuscator for* P/poly*, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}VI)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}VII)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The proof of Claim A.39 employs the same type of logic as that applied in Claim A.29 and hence we do not provide the details in this case as well. □

Claim A.40. *Assuming* ACC *is a positional accumulator satisfying the 'indistinguishability of read setup' property defined in Definition 2.4, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}VII)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-}VIII)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-VII})}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4\text{-VIII})}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the indistinguishability of read setup property of the positional accumulator ACC using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Then, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, then $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. On the other hand, if $b = 1$, then it chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
  4. $\mathcal{B}$ returns the challenge CPRF value y^* to $\mathcal{A}$.
- For $\eta \in [\hat{q}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, if $\eta \neq \nu$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{0,\nu-1,4\text{-VII}}$, while if $\eta = \nu$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ selects PPRF keys $K_1^{(\nu)}, \ldots, K_\lambda^{(\nu)}, K_{\text{SPS},A}^{(\nu)}, K_{\text{SPS},B}^{(\nu)}, K_{\text{SPS},E}^{(\nu)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Then, it forms the punctured PPRF keys $K_{\text{SPS},A}^{(\nu)}\{(h^*, \ell^*, 0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},A}^{(\nu)}, (h^*, \ell^*, 0))$ and $K_{\text{SPS},B}^{(\nu)}\{(h^*, \ell^*, 0)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{SPS},B}^{(\nu)}, (h^*, \ell^*, 0))$.
  3. Next, $\mathcal{B}$ sends $n_{\text{ACC-BLK}} = 2^\lambda$, the sequence of symbol-index pairs $((x_0^*, 0), \ldots, (x_{\ell^*-1}^*, \ell^*-1))$, and the index $i^* = 0$ to its ACC read setup indistinguishability challenger $\mathcal{C}$ and receives back $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0)$, where either $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ or $(\text{PP}_{\text{ACC}}, w_0, \text{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x_0^*, 0), \ldots, (x_{\ell^*-1}^*, \ell^* - 1)), i^* = 0)$.
  4. Next, it generates $(\text{PP}_{\text{ITR}}^{(\nu)}, v_0^{(\nu)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  5. Then, it sets $m_{0,0}^{(\nu)} = (v_0^{(\nu)}, q_0^{(\nu)}, w_0, 0)$. For $j = 1, \ldots, \ell^*$, it iteratively computes the following:
     - $\text{AUX}_j^{(\nu)} = \mathsf{ACC.Prep\text{-}Write}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1)$
     - $w_j = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{j-1}, x_{j-1}^*, j-1, \text{AUX}_j^{(\nu)})$
     - $\text{STORE}_j = \mathsf{ACC.Write\text{-}Store}(\text{PP}_{\text{ACC}}, \text{STORE}_{j-1}, j-1, x_{j-1}^*)$
     - $v_j^{(\nu)} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}^{(\nu)}, v_{j-1}^{(\nu)}, (q_0^{(\nu)}, w_{j-1}, 0))$

It sets $m^{(\nu)}_{\ell^*,0} = (v^{(\nu)}_{\ell^*}, q^{(\nu)}_0, w_{\ell^*}, 0)$.

6. After that, it generates $(\mathrm{SK}^{(\nu,0)}_{\mathrm{SPS},C}, \mathrm{VK}^{(\nu,0)}_{\mathrm{SPS},C}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-REJ},C}), (\mathrm{SK}^{(\nu,0)}_{\mathrm{SPS},D}, \mathrm{VK}^{(\nu,0)}_{\mathrm{SPS},D}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-REJ},D}) \xleftarrow{\$} \mathsf{SPS.Setup}(1^\lambda)$ and forms $(\sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{\ell^*,0},C}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},C}, \mathrm{SK}^{(\nu,0)}_{\text{SPS-ABO},C}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ABO},C}) \xleftarrow{\$} \mathsf{SPS.Split}(\mathrm{SK}^{(\nu,0)}_{\mathrm{SPS},C}, m^{(\nu)}_{\ell^*,0})$, $(\sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{\ell^*,0},D}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},D}, \mathrm{SK}^{(\nu,0)}_{\text{SPS-ABO},D}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ABO},D}) \xleftarrow{\$} \mathsf{SPS.Split}(\mathrm{SK}^{(\nu,0)}_{\mathrm{SPS},D}, m^{(\nu)}_{\ell^*,0})$.
7. It gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CPRF}}\{M^{(\nu)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, v^{(\nu)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\nu)}_0, w_0, v^{(\nu)}_0, K^{(\nu)}_{\mathrm{SPS},E}]), \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K^{(\nu)}_{\mathrm{SPS},E}]), \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}^{(4,2)}[K^{(\nu)}_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\mathrm{SPS},E}, \\ \quad \sigma^{(\nu,0)}_{\text{SPS-ONE},m^{(\nu)}_{\ell^*,0},C}, \mathrm{SK}^{(\nu,0)}_{\text{SPS-ABO},D}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]), \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}^{(1,0,2)}_{\mathrm{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, \\ \quad K^{(\nu)}_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},C}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ABO},D}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its ACC read setup indistinguishability experiment.

Note that if $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,\text{4-VII}}$. On the other hand, if $(\mathrm{PP}_{\mathrm{ACC}}, w_0, \mathrm{STORE}_0) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x^*_0, 0), \ldots, (x^*_{\ell^*-1}, \ell^*-1)), i^* = 0)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,\nu-1,\text{4-VIII}}$. This completes the proof of Claim A.40. □

Claim A.41. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$ *and* ACC *is a positional accumulator satisfying the 'read enforcing' property defined in Definition 2.4, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ,* $|\mathsf{Adv}^{(0,\nu-1,\text{4-}VIII)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,\text{4-}IX)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. The only difference between $\mathsf{Hyb}_{0,\nu-1,\text{4-VIII}}$ and $\mathsf{Hyb}_{0,\nu-1,\text{4-IX}}$ is the following:
In $\mathsf{Hyb}_{0,\nu-1,\text{4-VIII}}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0)$ within the ν^{th} constrained key provided to $\mathcal{A}$, while in $\mathsf{Hyb}_{0,\nu-1,\text{4-IX}}$ it includes the program $\mathcal{IO}(P_1)$ instead, where

- $P_0 = \mathsf{Constrained\text{-}Key.Prog}^{(1,0,2)}_{\mathrm{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},C}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ABO},D}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]$ (Fig. A.34),
- $P_1 = \mathsf{Constrained\text{-}Key.Prog}^{(1,0,3)}_{\mathrm{CPRF}}[M^{(\nu)}, T = 2^\lambda, t^{*(\nu)}, \mathrm{PP}^{(\nu)}_{\mathrm{ACC}}, \mathrm{PP}^{(\nu)}_{\mathrm{ITR}}, K, K^{(\nu)}_1, \ldots, K^{(\nu)}_\lambda, K^{(\nu)}_{\mathrm{SPS},A}\{(h^*,\ell^*,0)\}, K^{(\nu)}_{\mathrm{SPS},B}\{(h^*,\ell^*,0)\}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},C}, \mathrm{VK}^{(\nu,0)}_{\text{SPS-ABO},D}, m^{(\nu)}_{\ell^*,0}, h^*, \ell^*]$ (Fig. A.35).

We will argue that the programs P_0 and P_1 are functionally equivalent, so that, by the security of $\mathcal{IO}$ Claim A.41 follows. Clearly, the only inputs for which the outputs of the two programs might differ are those corresponding to $(h, \ell_{\mathrm{INP}}, t) = (h^*, \ell^*, 1)$. For inputs corresponding to $(h^*, \ell^*, 1)$, P_1 is programmed to output $\perp$ in case $\mathrm{ST}_{\mathrm{OUT}} = q_{\mathrm{AC}}$ but $\alpha =$ 'A', whereas, P_0 has no such condition in its programming. Now, observe that for inputs corresponding to $(h^*, \ell^*, 1)$, both the programs will assign the value 'A' to α if and only if $\mathsf{SPS.Verify}(\mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},C}, m_{\mathrm{IN}}, \sigma_{\mathrm{SPS,IN}}) = 1$, where $\mathrm{VK}^{(\nu,0)}_{\text{SPS-ONE},C}$ is generated by running $\mathsf{SPS.Split}(\mathrm{SK}^{(\nu,0)}_{\mathrm{SPS},C}, m^{(\nu)}_{\ell^*,0})$. Hence, by the correctness [Properties (i), (iii) and (v)] of the splittable signature scheme SPS, described in Definition 2.6, it is immediate that for inputs corresponding to $(h^*, \ell^*, 1)$, both programs will

set $\alpha =$'A' only if $m_{\text{IN}} = m^{(\nu)}_{\ell^*,0}$. Now, $m_{\text{IN}} = m^{(\nu)}_{\ell^*,0}$ means $\text{ST}_{\text{IN}} = q_0^{(\nu)}, w_{\text{IN}} = w^{(\nu)}_{\ell^*}$, and $\text{POS}_{\text{IN}} = 0$. Further, recall that in both the hybrid experiments $\mathsf{Hyb}_{0,\nu-1,4\text{-VIII}}$ and $\mathsf{Hyb}_{0,\nu-1,4\text{-IX}}$, $(\text{PP}^{(\nu)}_{\text{ACC}}, w_0^{(\nu)}, \text{STORE}_0^{(\nu)}) \xleftarrow{\$} \mathsf{ACC.Setup\text{-}Enforce\text{-}Read}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda, ((x_0^*, 0), \ldots, (x^*_{\ell^*-1}, \ell^* - 1)), i^* = 0)$. Therefore, by the read enforcing property of ACC it follows that if $w_{\text{IN}} = w^{(\nu)}_{\ell^*}$ and $\text{POS}_{\text{IN}} = 0$, then $\mathsf{ACC.Verify\text{-}Read}(\text{PP}^{(\nu)}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 1$ implies $\text{SYM}_{\text{IN}} = x_0^*$. Hence, for both the programs, for inputs corresponding to $(h^*, \ell^*, 1)$, $\alpha =$'A' implies $\text{ST}_{\text{IN}} = q_0^{(\nu)}$ and $\text{SYM}_{\text{IN}} = x_0^*$, which in turn implies $\text{ST}_{\text{OUT}} \neq q_{\text{AC}}$. Hence, the two programs have identical outputs for inputs corresponding to $(h^*, \ell^*, 1)$ as well. Thus, the two programs are functionally equivalent. □

Claim A.42. *Assuming* ACC *is a positional accumulator satisfying the 'indistinguishability of read setup' property defined in Definition 2.4, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}^{(0,\nu-1,4\text{-}IX)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4\text{-}X)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. The proof of Claim A.42 is similar to that of Claim A.40 with some appropriate modifications that are easy to figure out. □

Claim A.43. *Assuming* SPS *is a splittable signature scheme satisfying 'splitting indistinguishability' as per Definition 2.6, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}^{(0,\nu-1,4\text{-}X)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4\text{-}XI)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. The proof of Claim A.43 proceeds along a similar path as that of the proof of Claim A.30. We omit the details here. □

Claim A.44. *Assuming* $\mathcal{IO}$ *is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$*, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}^{(0,\nu-1,4\text{-}XI)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4\text{-}XII)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. The proof of Claim A.44 employs a similar type of logic as that utilized in the proof of Claim A.31. We omit the details in this case as well. □

Claim A.45. *Assuming* $\mathcal{F}$ *is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}^{(0,\nu-1,4\text{-}XII)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4\text{-}XIII)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. The proof of Claim A.45 takes an analogous path as that taken by the proof of Claim A.25. The details are omitted. □

Claim A.46. *Assuming* $\mathcal{IO}$ *is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$ *and* $\mathcal{F}$ *satisfies the correctness under puncturing property defined in Definition 2.2, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}^{(0,\nu-1,4\text{-}XIII)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4\text{-}XIV)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. The proof of Claim A.46 applies the same kind of logic as that employed in the proof of Claim A.24. The details are again omitted. □

□

Lemma A.12. *Assuming* $\mathcal{IO}$ *is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$*,* ACC *is a secure positional accumulator according to Definition 2.4, and* ITR *is a secure cryptographic iterator as per Definition 2.5, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}^{(0,\nu-1,4,(\gamma-1)')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu-1,4,\gamma)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. The proof of Lemma A.12 follows an analogous path to that of Lemma B.4 of [DKW16] with certain appropriate modifications that are easy to identify. Observe that now the 'indistinguishability of read setup' and 'indistinguishability of write setup' properties of ACC can be safely employed for the transition of hybrids as separate ACC public parameters have been associated with distinct constrained keys. We omit the details to avoid repetition. □

Lemma A.13. *Assuming $\mathcal{IO}$ is a a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* ACC *is a positional accumulator possessing the 'indistinguishability of read setup' as well as 'read enforcing' properties defined in Definition 2.4, and* SPS *is a splitable signature scheme satisfying '$\text{VK}_{\text{SPS-ONE}}$ indistinguishability', '$\text{VK}_{\text{SPS-ABO}}$ indistinguishability', as well as 'splitting indistinguishability' properties as defined in Definition 2.6, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4,\gamma)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4,\gamma')}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of Lemma A.13 is similar to that of Lemma B.3 of [DKW16] with some appropriate readily identifiable changes. Note that now the 'indistinguishability of read setup' property of ACC can be safely utilized for the transition as separate ACC public parameters have been associated with different constrained keys. Here again we omit the details to avoid repetition. □

Lemma A.14. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and ACC is a positional accumulator having 'indistinguishability of read setup' and 'read enforcing' properties defined in Definition 2.4, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,4,(t^{*(\nu)}-1)')}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,5)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of Lemma A.14 is similar to that of Lemma B.5 of [DKW16] with some appropriate changes that are easy to determine. Here also the 'indistinguishability of read setup' property of ACC can be safely used due to the assignment of distinct ACC public parameters with different constrained keys. The details are omitted again to avoid repetition. □

Lemma A.15. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, SPS is a splitable signature scheme possessing the '$\text{VK}_{\text{SPS-REJ}}$ indistinguishability' property, and PRG is a secure injective pseudorandom generator, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,5)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1,3,6)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl*.*

Proof. The proof of Lemma A.15 is similar to that of Lemma B.6 of [DKW16]. □

Appendix B: Proof of Theorem 4.1

Theorem 4.1 (Security of the CVPRF Construction of Section 4.2). *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a secure positional accumulator as defined in Definition 2.4,* ITR *is a secure cryptographic iterator as per Definition 2.5,* SPS *is a secure splittable signature scheme according to Definition 2.6,* PRG *is a secure injective pseudorandom generator, and* PKE *is a perfectly correct* CPA *secure public key encryption scheme, the* CVPRF *construction described in Section 4.2 satisfies all the properties of a secure* CVPRF *defined in Definition 4.1 of Section 4.1.*

Proof.

► **Provability**: The provability of the CVPRF construction of Section 4.2 follows directly from its construction.

► **Uniqueness**: The uniqueness property follows from the perfect correctness property of the PKE scheme. More precisely, assume that there exists a CVPRF public verification key $\text{VK}_{\text{CVPRF}} = (\text{HK}, \mathcal{V}_{\text{CVPRF}})$, input $x \in \mathcal{X}_{\text{CVPRF}}$, and two value-proof pairs $(y_0, \pi_{\text{CVPRF},0} = (\text{PK}_{\text{PKE},0}, r_0))$, $(y_1, \pi_{\text{CVPRF},1} = (\text{PK}_{\text{PKE},1}, r_1)) \in \mathcal{Y}_{\text{CVPRF}} \times \Pi_{\text{CVPRF}}$ such that

$$
\begin{aligned}
[y_0 \neq y_1] \ \wedge \ & [\mathsf{CVPRF.Verify}(\text{VK}_{\text{CVPRF}}, x, y_0, \pi_{\text{CVPRF},0}) = 1] \wedge \\
& [\mathsf{CVPRF.Verify}(\text{VK}_{\text{CVPRF}}, x, y_1, \pi_{\text{CVPRF},1}) = 1], \\
\text{i.e.,} [y_0 \neq y_1] \ \wedge \ & [(\text{PK}_{\text{PKE},0} = \widehat{\text{PK}}_{\text{PKE}}) \ \wedge \ (\mathsf{PKE.Encrypt}(\text{PK}_{\text{PKE},0}, y_0; r_0) = \widehat{\text{CT}}_{\text{PKE}})] \wedge \\
& [(\text{PK}_{\text{PKE},1} = \widehat{\text{PK}}_{\text{PKE}}) \ \wedge \ (\mathsf{PKE.Encrypt}(\text{PK}_{\text{PKE},1}, y_1; r_1) = \widehat{\text{CT}}_{\text{PKE}})] \\
& \text{where } (\widehat{\text{PK}}_{\text{PKE}}, \widehat{\text{CT}}_{\text{PKE}}) = \mathcal{V}_{\text{CVPRF}}(\mathcal{H}_{\text{HK}}(x), |x|), \\
\text{i.e.,} [y_0 \neq y_1] \ \wedge \ & [\mathsf{PKE.Encrypt}(\widehat{\text{PK}}_{\text{PKE}}, y_0; r_0) = \mathsf{PKE.Encrypt}(\widehat{\text{PK}}_{\text{PKE}}, y_1; r_1)],
\end{aligned}
$$

which contradicts the perfect correctness property of PKE.

► **Constraint Hiding**: The constraint hiding property is also obvious. For any input $x \in \mathcal{X}_{\text{CVPRF}}$, the proof $\pi_{\text{CVPRF}} \in \Pi_{\text{CVPRF}}$ generated by $\mathsf{CVPRF.Prove}(\text{SK}_{\text{CVPRF}}, x)$ is $\pi_{\text{CVPRF}} = (\text{PK}_{\text{PKE}}, r_{\text{PKE},2})$, where $(\text{PK}_{\text{PKE}}, \text{SK}_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda; r_{\text{PKE},1})$ and $r_{\text{PKE},1} \| r_{\text{PKE},2} = \mathcal{F}(K_{\text{PKE}}, (\mathcal{H}_{\text{HK}}(x), |x|))$, $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ being generated during the setup. Observe that this value of π_{CVPRF} is exactly the same as outputted by $\mathsf{CVPRF.Prove\text{-}Constrained}(\text{SK}_{\text{CVPRF}}\{M\}, x)$ for any TM $M \in \mathbb{M}_\lambda$ with $M(x) = 1$.

► **Selective Pseudorandomness**: We will prove selective pseudorandomness of the CVPRF construction of Section 4.2 through a sequence of hybrid experiments by arguing depending on the security of various primitives that the advantage of any PPT adversary $\mathcal{A}$ in consecutive hybrid experiments differ only negligibly as well as that in the final hybrid experiment is negligible. Just like the proof of Theorem 3.1, here also working in the selective model ensures that the challenger $\mathcal{B}$ knows the challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CVPRF}}$ and the SSB hash value $h^* = \mathcal{H}_{\text{HK}}(x^*)$ prior to receiving any constrained key query from the adversary $\mathcal{A}$. We assume that the total number of constrained key queries made by the adversary $\mathcal{A}$ is $\hat{q}$. In view of Remark 4.1, we do not consider any proof query. The description of hybrid experiments follows:

Sequence of Hybrid Experiments

Hyb_0: This experiment corresponds to the real selective pseudorandomness experiment described in Definition 4.1. More precisely, this experiment proceeds as follows:

- $\mathcal{A}$ submits a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CVPRF}}$ with $|x^*| = \ell^*$ to $\mathcal{B}$.
- $\mathcal{B}$ generates $(\text{SK}_{\text{CVPRF}} = (K, K_{\text{PKE}}, \text{HK}), \text{VK}_{\text{CVPRF}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{CVPRF}}[K, K_{\text{PKE}}])) \xleftarrow{\$} \mathsf{CVPRF.Setup}(1^\lambda)$ as described in the Section 4.2. Next, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathsf{CVPRF}(\text{SK}_{\text{CVPRF}}, x^*) = \mathcal{F}(K, (h^* = \mathcal{H}_{\text{HK}}(x^*), \ell^*))$. Otherwise, $\mathcal{B}$ chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{CVPRF}}$. $\mathcal{B}$ provides $\mathcal{A}$ with $(\text{VK}_{\text{CVPRF}}, y^*)$.

- For $\eta = 1, \ldots, \hat{q}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} = \langle Q^{(\eta)}, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta^{(\eta)}, q_0^{(\eta)}, q_{\text{AC}}^{(\eta)}, q_{\text{REJ}}^{(\eta)} \rangle \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ creates

$$\text{SK}_{\text{CVPRF}}\{M^{(\eta)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CVPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K, K_{\text{PKE}}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, \\ \quad K_{\text{SPS},A}^{(\eta)}]) \end{array} \right) \xleftarrow{\$} \mathsf{CVPRF.Constrain}(\text{SK}_{\text{CVPRF}}, M^{(\eta)}),$$

  as described in Section 4.2, and returns $\text{SK}_{\text{CVPRF}}\{M^{(\eta)}\}$ to $\mathcal{A}$. Here, we assign the index η to all the components which are generated during the execution of $\mathsf{CVPRF.Constrain}(\text{SK}_{\text{CVPRF}}, M^{(\eta)})$.
- Eventually, $\mathcal{A}$ outputs a guess bit $b' \in \{0, 1\}$.

$\mathsf{Hyb}_{0,\nu}$ ($\nu = 1, \ldots, \hat{q}$): This experiment is similar to Hyb_0 except that for $\eta \in [\hat{q}]$, in reply to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ returns the constrained key

$$\text{SK}_{\text{CVPRF}}\{M^{(\eta)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}'_{\text{CVPRF}}}[M^{(\eta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K, K_{\text{PKE}}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\text{SPS},A}^{(\eta)}, \\ \quad \underline{h^*, \ell^*}]) \end{array} \right),$$

if $\underline{\eta \leq \nu}$, where the program $\underline{\mathsf{Constrained\text{-}Key.Prog}'_{\text{CVPRF}}}$ is a modification of the program $\mathsf{Constrained\text{-}Key.Prog}_{\text{CVPRF}}$ (Fig. 4.2) and is described in Fig. B.1, while it returns the constrained key

$$\text{SK}_{\text{CVPRF}}\{M^{(\eta)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{CVPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K, K_{\text{PKE}}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\text{SPS},A}^{(\eta)}]) \end{array} \right),$$

if $\underline{\eta > \nu}$. Observe that $\mathsf{Hyb}_{0,0}$ coincides with Hyb_0.

Hyb_1: This experiment coincides with $\mathsf{Hyb}_{0,\hat{q}}$. More formally, in this experiment for $\eta = 1, \ldots, \hat{q}$, in reply to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ generates all the components of the constrained key as in Hyb_0, however, it returns the

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_{\text{PKE}}, K_1, \ldots, K_\lambda, K_{\text{SPS},A}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: (CVPRF evaluation $\mathcal{F}(K, (h, \ell_{\text{INP}}))$, CVPRF proof $\pi_{\text{CVPRF}} = (\text{PK}_{\text{PKE}}, r_{\text{PKE},2})$) or Header Position ($\text{POS}_{\text{OUT}}$, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SSB.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   (b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \ \wedge \ [(h, \ell_{\text{INP}}) \neq (h^*, \ell^*)]$, perform the following:
   (I) Compute $r_{\text{PKE},1} \| r_{\text{PKE},2} = \mathcal{F}(K_{\text{PKE}}, (h, \ell_{\text{INP}}))$, $(\text{PK}_{\text{PKE}}, \text{SK}_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda;\ r_{\text{PKE},1})$.
   (II) Output $(\mathcal{F}(k, (h, \ell_{\text{INP}})), \pi_{\text{CVPRF}} = (\text{PK}_{\text{PKE}}, r_{\text{PKE},2}))$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, output $\perp$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. B.1. $\mathsf{Constrained\text{-}Key.Prog}'_{\text{CVPRF}}$

constrained key as

$$\text{SK}_{\text{CVPRF}}\{M^{(\eta)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)} v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\text{CVPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K, K_{\text{PKE}}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\text{SPS},A}^{(\eta)}, \\ \quad h^*, \ell^*]) \end{array} \right).$$

The rest of the experiment is analogous to Hyb_0.

Hyb_2: This experiment is analogous to Hyb_1 other than the following exceptions:

(I) Upon receiving the challenge input x^*, $\mathcal{B}$ proceeds as follows:
1. It first selects $K, K_{\text{PKE}} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$ and generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ just as in Hyb_1.
2. It then computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$ and creates the punctured PPRF key $K_{\text{PKE}}\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{PKE}}, (h^*, \ell^*))$.
3. Next, it computes $\widehat{r}^*_{\text{PKE},1} \| \widehat{r}^*_{\text{PKE},2} = \mathcal{F}(K_{\text{PKE}}, (h^*, \ell^*))$, forms $(\widehat{\text{PK}}^*_{\text{PKE}}, \widehat{\text{SK}}^*_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda; \widehat{r}^*_{\text{PKE},1})$ and computes $\widehat{\text{CT}}^*_{\text{PKE}} = \mathsf{PKE.Encrypt}(\widehat{\text{PK}}^*_{\text{PKE}}, \mathcal{F}(K, (h^*, \ell^*));\ \widehat{r}^*_{\text{PKE},2})$.
4. It sets the public verification key VK_{CVPRF} to be given to $\mathcal{A}$ as $\text{VK}_{\text{CVPRF}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\text{CVPRF}}[K, K_{\text{PKE}}\{(h^*, \ell^*)\}, \widehat{\text{PK}}^*_{\text{PKE}}, \widehat{\text{CT}}^*_{\text{PKE}}, h^*, \ell^*]))$, where the program

$\mathsf{Verify.Prog}'_{\mathrm{CVPRF}}$ is an alteration of the program $\mathsf{Verify.Prog}_{\mathrm{CVPRF}}$ (Fig. 4.1) and is depicted in Fig. B.2.

Constants: PPRF key K, Punctured PPRF key $K_{\mathrm{PKE}}\{(h^*, \ell^*)\}$, PKE public key $\widehat{\mathrm{PK}}^*_{\mathrm{PKE}}$, PKE ciphertext $\widehat{\mathrm{CT}}^*_{\mathrm{PKE}}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*
Inputs: SSB hash value h, Length ℓ_{INP}
Output: (PKE public key $\widehat{\mathrm{PK}}_{\mathrm{PKE}}$, Encryption of CVPRF value $\widehat{\mathrm{CT}}_{\mathrm{PKE}}$)

(a) If $(h, \ell_{\mathrm{INP}}) = (h^*, \ell^*)$, output $(\widehat{\mathrm{PK}}^*_{\mathrm{PKE}}, \widehat{\mathrm{CT}}^*_{\mathrm{PKE}})$.
Else, compute $\hat{r}_{\mathrm{PKE},1} \| \hat{r}_{\mathrm{PKE},2} = \mathcal{F}(K_{\mathrm{PKE}}\{(h^*, \ell^*\}, (h, \ell_{\mathrm{INP}}))$, $(\widehat{\mathrm{PK}}_{\mathrm{PKE}}, \widehat{\mathrm{SK}}_{\mathrm{PKE}}) = \mathsf{PKE.Setup}(1^\lambda;\ \hat{r}_{\mathrm{PKE},1})$.
(b) Compute $\widehat{\mathrm{CT}}_{\mathrm{PKE}} = \mathsf{PKE.Encrypt}(\widehat{\mathrm{PK}}_{\mathrm{PKE}}, \mathcal{F}(K, (h, \ell_{\mathrm{INP}}));\ \hat{r}_{\mathrm{PKE},2})$.
(c) Output $(\widehat{\mathrm{PK}}_{\mathrm{PKE}}, \widehat{\mathrm{CT}}_{\mathrm{PKE}})$.

Fig. B.2. $\mathsf{Verify.Prog}'_{\mathrm{CVPRF}}$

(II) For $\eta = 1, \ldots, \hat{q}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ generates all the components as in Hyb_1, however, provides $\mathcal{A}$ with the constrained key

$$\mathrm{SK}_{\mathrm{CVPRF}}\{M^{(\eta)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}^{(\eta)}_{\mathrm{ACC}}, w^{(\eta)}_0, \mathrm{STORE}^{(\eta)}_0, \mathrm{PP}^{(\eta)}_{\mathrm{ITR}}, v^{(\eta)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\eta)}_0, w^{(\eta)}_0, v^{(\eta)}_0, K^{(\eta)}_{\mathrm{SPS},E}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}^{(\eta)}_{\mathrm{ACC}}, \mathrm{PP}^{(\eta)}_{\mathrm{ITR}}, K^{(\eta)}_{\mathrm{SPS},E}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\eta)}_{\mathrm{SPS},A}, K^{(\eta)}_{\mathrm{SPS},E}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CVPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}^{(\eta)}_{\mathrm{ACC}}, \mathrm{PP}^{(\eta)}_{\mathrm{ITR}}, K, \underline{K_{\mathrm{PKE}}\{(h^*, \ell^*)\}}, \\ \qquad K^{(\eta)}_1, \ldots, K^{(\eta)}_\lambda, K^{(\eta)}_{\mathrm{SPS},A}, h^*, \ell^*]) \end{array}\right).$$

Hyb_3: This experiment is identical to Hyb_2 except that $\mathcal{B}$ selects $\underline{\hat{r}^*_{\mathrm{PKE},1} \| \hat{r}^*_{\mathrm{PKE},2} \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}}$. More formally, $\mathcal{B}$ computes the punctured PPRF key $K_{\mathrm{PKE}}\{(h^*, \ell^*)\}$ as before, however, it generates $\underline{(\widehat{\mathrm{PK}}^*_{\mathrm{PKE}}, \widehat{\mathrm{SK}}^*_{\mathrm{PKE}}) \xleftarrow{\$} \mathsf{PKE.Setup}(1^\lambda)}$ and computes $\underline{\widehat{\mathrm{CT}}^*_{\mathrm{PKE}} \xleftarrow{\$} \mathsf{PKE.Encrypt}(\widehat{\mathrm{PK}}^*_{\mathrm{PKE}}, \mathcal{F}(K, (h^*, \ell^*)))}$. $\mathcal{B}$ gives $\mathrm{VK}_{\mathrm{CVPRF}} = (\mathrm{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\mathrm{CVPRF}}[K, K_{\mathrm{PKE}}\{(h^*, \ell^*)\}, \widehat{\mathrm{PK}}^*_{\mathrm{PKE}}, \widehat{\mathrm{CT}}^*_{\mathrm{PKE}}, h^*, \ell^*]))$ to $\mathcal{A}$ as earlier.

Hyb_4: This experiment is the same as Hyb_3 with the only exception that $\widehat{\mathrm{CT}}^*_{\mathrm{PKE}}$ now encrypts a uniformly random value in $\mathcal{Y}_{\mathrm{PPRF}}$ rather than $\mathcal{F}(K, (h^*, \ell^*))$. More precisely, $\mathcal{B}$ generates $(\widehat{\mathrm{PK}}^*_{\mathrm{PKE}}, \widehat{\mathrm{SK}}^*_{\mathrm{PKE}}) \xleftarrow{\$} \mathsf{PKE.Setup}(1^\lambda)$ as in Hyb_3, however, it now chooses $\underline{\hat{y}^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}}$ and computes $\underline{\widehat{\mathrm{CT}}^*_{\mathrm{PKE}} \xleftarrow{\$} \mathsf{PKE.Encrypt}(\widehat{\mathrm{PK}}^*_{\mathrm{PKE}}, \hat{y}^*)}$. $\mathcal{B}$ provides $\mathcal{A}$ with $\mathrm{VK}_{\mathrm{CVPRF}} = (\mathrm{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\mathrm{CVPRF}}[K, K_{\mathrm{PKE}}\{(h^*, \ell^*)\}, \widehat{\mathrm{PK}}^*_{\mathrm{PKE}}, \widehat{\mathrm{CT}}^*_{\mathrm{PKE}}, h^*, \ell^*]))$ as before.

Hyb_5: This experiment is analogous to Hyb_4 except the following exceptions:

(I) Upon receiving the challenge input $x^* \in \mathcal{X}_{\mathrm{CVPRF}}$, $\mathcal{B}$ proceeds as follows:
1. it first picks PPRF keys $K, K_{\mathrm{PKE}} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$ and generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$.
2. Next, it computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$ and it creates the punctured PPRF keys $\underline{K\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K, (h^*, \ell^*))}$, $K_{\mathrm{PKE}}\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\mathrm{PKE}}, (h^*, \ell^*))$.
3. After that, it generates $(\widehat{\mathrm{PK}}^*_{\mathrm{PKE}}, \widehat{\mathrm{SK}}^*_{\mathrm{PKE}}) \xleftarrow{\$} \mathsf{PKE.Setup}(1^\lambda)$, selects $\hat{y}^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$, and computes $\widehat{\mathrm{CT}}^*_{\mathrm{PKE}} \xleftarrow{\$} \mathsf{PKE.Encrypt}(\widehat{\mathrm{PK}}^*_{\mathrm{PKE}}, \hat{y}^*)$.

4. It sets the public verification key $\mathrm{VK}_{\mathrm{CVPRF}}$ to be sent to $\mathcal{A}$ as $\mathrm{VK}_{\mathrm{CVPRF}} = (\mathrm{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\mathrm{CVPRF}}[\underline{K\{(h^*,\ell^*)\}}, K_{\mathrm{PKE}}\{(h^*,\ell^*)\}, \widehat{\mathrm{PK}}^*_{\mathrm{PKE}}, \widehat{\mathrm{CT}}^*_{\mathrm{PKE}}, h^*, \ell^*]))$.

(II) For $\eta = 1, \ldots, \hat{q}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ generates all the components as in Hyb_4, however, it provides $\mathcal{A}$ with the constrained key

$$\mathrm{SK}_{\mathrm{CVPRF}}\{M^{(\eta)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}^{(\eta)}_{\mathrm{ACC}}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}^{(\eta)}_{\mathrm{ITR}}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K^{(\eta)}_{\mathrm{SPS},E}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}^{(\eta)}_{\mathrm{ACC}}, \mathrm{PP}^{(\eta)}_{\mathrm{ITR}}, K^{(\eta)}_{\mathrm{SPS},E}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\eta)}_{\mathrm{SPS},A}, K^{(\eta)}_{\mathrm{SPS},E}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CVPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}^{(\eta)}_{\mathrm{ACC}}, \mathrm{PP}^{(\eta)}_{\mathrm{ITR}}, \underline{K\{(h^*,\ell^*)\}}, K_{\mathrm{PKE}}\{(h^*,\ell^*)\}, \\ \quad K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K^{(\eta)}_{\mathrm{SPS},A}, h^*, \ell^*]) \end{array}\right).$$

Analysis

Let $\mathsf{Adv}^{(0)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(0,\nu)}_{\mathcal{A}}(\lambda)$ $(\nu = 1, \ldots, \hat{q}), \mathsf{Adv}^{(1)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(4)}_{\mathcal{A}}(\lambda)$, and $\mathsf{Adv}^{(5)}_{\mathcal{A}}(\lambda)$ represent respectively the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in $\mathsf{Hyb}_0, \mathsf{Hyb}_{0,\nu}$ $(\nu = 1, \ldots, \hat{q}), \mathsf{Hyb}_1, \mathsf{Hyb}_2, \mathsf{Hyb}_3, \mathsf{Hyb}_4$, and Hyb_5 respectively. Then, by the description of the hybrid experiments it follows that $\mathsf{Adv}^{\mathrm{CVPRF,SEL\text{-}PR}}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,0)}_{\mathcal{A}}(\lambda)$ and $\mathsf{Adv}^{(1)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,\hat{q})}_{\mathcal{A}}(\lambda)$. Hence, we have

$$\mathsf{Adv}^{\mathrm{CVPRF,SEL\text{-}PR}}_{\mathcal{A}}(\lambda) \leq \sum_{\nu=1}^{\hat{q}} |\mathsf{Adv}^{(0,\nu-1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu)}_{\mathcal{A}}(\lambda)| + \sum_{j=1}^{4} |\mathsf{Adv}^{(j)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(j+1)}_{\mathcal{A}}(\lambda)| + \mathsf{Adv}^{(5)}_{\mathcal{A}}(\lambda). \tag{B.1}$$

Lemmas B.1–B.6 will show that the RHS of Eq. (B.1) is negligible and thus the selective pseudorandomness of the CVPRF construction of Section 4.2 follows. □

B.1 Lemmas for the Proof of Theorem 4.1

Lemma B.1. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a secure positional accumulator as defined in Definition 2.4,* ITR *is a secure cryptographic iterator as per Definition 2.5,* SPS *is a secure splittable signature scheme according to Definition 2.6, and PRG is a secure injective pseudorandom generator, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$, for some negligible function* negl.

Proof. The proof of Lemma B.1 is similar to that of Lemma A.3 and, therefore, is omitted to avoid repetition. □

Lemma B.2. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$ satisfies the correctness under puncturing property defined in Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$, for some negligible function* negl.

Proof. The two differences between Hyb_1 and Hyb_2 are the following:

(I) In Hyb_1, the challenger $\mathcal{B}$ includes the program $\mathcal{IO}(V_0)$ within $\mathrm{VK}_{\mathrm{CVPRF}}$, whereas, in Hyb_2, $\mathcal{B}$ includes the program $\mathcal{IO}(V_1)$, where

– $V_0 = \mathsf{Verify.Prog}_{\text{CVPRF}}[K, K_{\text{PKE}}]$ (Fig. 4.1),
– $V_1 = \mathsf{Verify.Prog}'_{\text{CVPRF}}[K, K_{\text{PKE}}\{(h^*, \ell^*)\}, \widehat{\text{PK}}^*_{\text{PKE}}, \widehat{\text{CT}}^*_{\text{PKE}}, h^*, \ell^*]$ (Fig. B.2).

(II) For $\eta = 1, \ldots, \hat{q}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ includes $\mathcal{IO}(P_1^{(\eta)})$ within $\text{SK}_{\text{CVPRF}}\{M^{(\eta)}\}$ in hyb_2 in place of $\mathcal{IO}(P_0^{(\eta)})$ in Hyb_1, where
– $P_0^{(\eta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\text{CVPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}^{(\eta)}_{\text{ACC}}, \text{PP}^{(\eta)}_{\text{ITR}}, K, K_{\text{PKE}}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K^{(\eta)}_{\text{SPS},A}, h^*, \ell^*]$ (Fig. B.1),
– $P_1^{(\eta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\text{CVPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}^{(\eta)}_{\text{ACC}}, \text{PP}^{(\eta)}_{\text{ITR}}, K, K_{\text{PKE}}\{(h^*, \ell^*)\}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K^{(\eta)}_{\text{SPS},A}, h^*, \ell^*]$ (Fig. B.1).

Now, observe that on input $(h, \ell_{\text{INP}}) \neq (h^*, \ell^*)$, both the programs V_0 and V_1 operates in the same manner only that the latter one uses the punctured PPRF key $K_{\text{PKE}}\{(h^*, \ell^*)\}$ for computing the string $\hat{r}_{\text{PKE},1} \| \hat{r}_{\text{PKE},2}$ instead of the full PPRF key K_{PKE} used by the former program. Therefore, by the correctness under puncturing property of PPRF $\mathcal{F}$, it follows that for all inputs $(h, \ell_{\text{INP}}) \neq (h^*, \ell^*)$, both the programs have identical output. Moreover, on input (h^*, ℓ^*), V_1 outputs the hardwired values $(\widehat{\text{PK}}^*_{\text{PKE}}, \widehat{\text{CT}}^*_{\text{PKE}})$ which are computed in Hyb_2 as $(\widehat{\text{PK}}^*_{\text{PKE}}, \widehat{\text{SK}}^*_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda;\ \hat{r}^*_{\text{PKE},1}), \widehat{\text{CT}}^*_{\text{PKE}} = \mathsf{PKE.Encrypt}(\widehat{\text{PK}}^*_{\text{PKE}}, \mathcal{F}(K, (h^*, \ell^*));\ \hat{r}^*_{\text{PKE},2})$, where $\hat{r}^*_{\text{PKE},1} \| \hat{r}^*_{\text{PKE},2} = \mathcal{F}(K_{\text{PKE}}, (h^*, \ell^*))$. Notice that these values are exactly the same as those outputted by V_0 on input (h^*, ℓ^*). Thus the two programs are functionally equivalent.

Further, note that the program $\mathsf{Constrained\text{-}Key.Prog}'_{\text{CVPRF}}$ computes $\mathcal{F}(K_{\text{PKE}}, (h, \ell_{\text{INP}}))$ if and only if $(h, \ell_{\text{INP}}) \neq (h^*, \ell^*)$. Thus, again by the correctness under puncturing property of PPRF $\mathcal{F}$, the programs $P_0^{(\eta)}$ and $P_1^{(\eta)}$ are functionally equivalent as well for all $\eta \in [\hat{q}]$.

Thus by the security of $\mathcal{IO}$ Lemma B.2 follows. Observe that to prove this lemma we would actually have to proceed through a sequence of intermediate hybrid experiments where in each hybrid experiment we switch the programs one at a time. □

Lemma B.3. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x^*_0 \ldots x^*_{\ell^*-1} \in \mathcal{X}_{\text{CVPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. $\mathcal{B}$ sends (h^*, ℓ^*) as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back a punctured PPRF key $K^*\{(h^*, \ell^*)\}$ and a value $r^* = r^*_1 \| r^*_2 \in \mathcal{Y}_{\text{PPRF}}$, where either $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$ or $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$. $\mathcal{B}$ *implicitly* views the key K^* as the key K_{PKE}.
  3. $\mathcal{B}$ then chooses $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  4. Next, it computes $(\widehat{\text{PK}}^*_{\text{PKE}}, \widehat{\text{SK}}^*_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda;\ r^*_1), \widehat{\text{CT}}^*_{\text{PKE}} = \mathsf{PKE.Encrypt}(\widehat{\text{PK}}^*_{\text{PKE}}, \mathcal{F}(K, (h^*, \ell^*));\ r^*_2)$.
  5. $\mathcal{B}$ sets $\text{VK}_{\text{CVPRF}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\text{CVPRF}}[K, K^*\{(h^*, \ell^*)\}, \widehat{\text{PK}}^*_{\text{PKE}}, \widehat{\text{CT}}^*_{\text{PKE}}, h^*, \ell^*]))$.
  6. $\mathcal{B}$ then selects a random bit $b \xleftarrow{\$} \{0, 1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. Otherwise, $\mathcal{B}$ chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
  7. $\mathcal{B}$ provides $(\text{VK}_{\text{CVPRF}}, y^*)$ to $\mathcal{A}$.

- For $\eta = 1, \ldots, \hat{q}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. After that, it generates $(\text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\eta)}, v_0^{(\eta)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CVPRF}}\{M^{(\eta)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\text{CVPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K, K^*\{(h^*, \ell^*)\}, \\ \qquad K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\text{SPS},A}^{(\eta)}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its PPRF selective pseudorandomness experiment.

Notice that if $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$, then $\mathcal{B}$ perfectly simulates Hyb_2. On the other hand, if $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, then $\mathcal{B}$ perfectly simulates hyb_3. This completes the proof of Lemma B.3. □

Lemma B.4. *Assuming* PKE *is* CPA *secure, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $|\mathsf{Adv}_{\mathcal{A}}^{(3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(4)}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(3)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(4)}(\lambda)|$ in non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the CPA security of PKE.

- $\mathcal{B}$ receives a PKE public key PK_{PKE}^* from its PKE CPA security challenger $\mathcal{C}$. $\mathcal{B}$ then initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\text{CVPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- After receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. $\mathcal{B}$ then selects PPRF keys $K, K_{\text{PKE}} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$ and creates the punctured PPRF key $K_{\text{PKE}}\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{PKE}}, (h^*, \ell^*))$.
  3. Next, $\mathcal{B}$ chooses $\hat{y}^* \in \mathcal{Y}_{\text{PPRF}}$. It sends $(\mathsf{MSG}_0 = \mathcal{F}(K, (h^*, \ell^*)), \mathsf{MSG}_1 = \hat{y}^*)$ as the pair of challenge messages to $\mathcal{C}$ and receives back a ciphertext CT_{PKE}^* from $\mathcal{C}$.
  4. $\mathcal{B}$ then sets the public verification key $\text{VK}_{\text{CVPRF}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\text{CVPRF}}[K, K_{\text{PKE}}\{(h^*, \ell^*)\}, \text{PK}_{\text{PKE}}^*, \text{CT}_{\text{PKE}}^*, h^*, \ell^*]))$.
  5. After that, $\mathcal{B}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. Otherwise, $\mathcal{B}$ chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
  6. $\mathcal{B}$ provides $\mathcal{A}$ with $(\text{VK}_{\text{CVPRF}}, y^*)$.
- For $\eta = 1, \ldots, \hat{q}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first chooses PPRF keys $K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. After that, it generates $(\text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\eta)}, v_0^{(\eta)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.

3. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\mathrm{SK}_{\mathrm{CVPRF}}\{M^{(\eta)}\} = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CVPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K_{\mathrm{PKE}}\{(h^*, \ell^*)\}, \\ \quad K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]) \end{array}\right).$$

- $\mathcal{A}$ eventually outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ in its PKE CPA security experiment.

Observe that if $\mathrm{CT}_{\mathrm{PKE}}^* \xleftarrow{\$} \mathsf{PKE.Encrypt}(\mathrm{PK}_{\mathrm{PKE}}^*, \mathcal{F}(K, (h^*, \ell^*)))$, then $\mathcal{B}$ perfectly simulates Hyb_3. On the other hand, if $\mathrm{CT}_{\mathrm{PKE}}^* \xleftarrow{\$} \mathsf{PKE.Encrypt}(\mathrm{PK}_{\mathrm{PKE}}^*, \hat{y}^*)$, then $\mathcal{B}$ perfectly simulates Hyb_4. This completes the proof Lemma B.4. □

Lemma B.5. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$ satisfies the correctness under puncturing property defined in Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(4)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(5)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. Hyb_4 and Hyb_5 differs with respect to the following:

(I) In Hyb_4 the challenger $\mathcal{B}$ includes the program $\mathcal{IO}(V_0)$ within the public verification key $\mathrm{VK}_{\mathrm{CVPRF}}$, while in Hyb_5, it includes the program $\mathcal{IO}(V_1)$ within $\mathrm{VK}_{\mathrm{CVPRF}}$, where
- $V_0 = \mathsf{Verify.Prog}'_{\mathrm{CVPRF}}[K, K_{\mathrm{PKE}}\{(h^*, \ell^*)\}, \widehat{\mathrm{PK}}_{\mathrm{PKE}}^*, \widehat{\mathrm{CT}}_{\mathrm{PKE}}^*, h^*, \ell^*]$,
- $V_1 = \mathsf{Verify.Prog}'_{\mathrm{CVPRF}}[K\{(h^*, \ell^*)\}, K_{\mathrm{PKE}}\{(h^*, \ell^*)\}, \widehat{\mathrm{PK}}_{\mathrm{PKE}}^*, \widehat{\mathrm{CT}}_{\mathrm{PKE}}^*, h^*, \ell^*]$,

the program $\mathsf{Verify.Prog}'_{\mathrm{CVPRF}}$ being depicted in Fig. B.2.

(II) For $\eta = 1, \ldots, \hat{q}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$, $\mathcal{B}$ provides $\mathcal{IO}(P_1^{(\eta)})$ within $\mathrm{SK}_{\mathrm{CVPRF}}\{M^{(\eta)}\}$ in the experiment Hyb_5 instead of including $\mathcal{IO}(P_0^{(\eta)})$ as in the experiment Hyb_4, where
- $P_0^{(\eta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CVPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K_{\mathrm{PKE}}\{(h^*, \ell^*)\}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]$,
- $P_1^{(\eta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CVPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K\{(h^*, \ell^*)\}, K_{\mathrm{PKE}}\{(h^*, \ell^*)\}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]$,

the program $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CVPRF}}$ being depicted in Fig. B.1.

Now, notice that the programs $\mathsf{Verify.Prog}'_{\mathrm{CVPRF}}$ and $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CVPRF}}$ compute $\mathcal{F}(K, (h, \ell_{\mathrm{INP}}))$ if and only if $(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)$. Hence, by the correctness under puncturing property of PPRF $\mathcal{F}$, it follows that the functionalities of the programs $\mathsf{Verify.Prog}'_{\mathrm{CVPRF}}$ and $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{CVPRF}}$ do not change if the punctured key $K\{(h^*, \ell^*)\}$ is hardwired in place of the full PPRF key K. Therefore, by the security of IO Lemma B.5 follows. Ofcourse, here again we would need to go through a sequence of intermediate hybrid experiments where in each hybrid experiment would change the hardwiring of one program at a time.

□

Lemma B.6. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $\mathsf{Adv}_{\mathcal{A}}^{((5)}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $\mathsf{Adv}^{(5)}_{\mathcal{A}}(\lambda)$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x^*_0 \ldots x^*_{\ell^*-1} \in \mathcal{X}_{\text{CVPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. $\mathcal{B}$ sends (h^*, ℓ^*) as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back a punctured PPRF key $K^*\{(h^*, \ell^*)\}$ and a value $r^* \in \mathcal{Y}_{\text{PPRF}}$, where either $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$ or $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$. $\mathcal{B}$ *implicitly* views the key K^* as the key K.
  3. $\mathcal{B}$ then chooses PPRF key $K_{\text{PKE}} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$ and creates the punctured PPRF key $K_{\text{PKE}}\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K_{\text{PKE}}, (h^*, \ell^*))$.
  4. Next, $\mathcal{B}$ generates $(\widehat{\text{PK}}^*_{\text{PKE}}, \widehat{\text{SK}}^*_{\text{PKE}}) \xleftarrow{\$} \mathsf{PKE.Setup}(1^\lambda)$, selects some $\hat{y}^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, and forms $\widehat{\text{CT}}^*_{\text{PKE}} \xleftarrow{\$} \mathsf{PKE.Encrypt}(\widehat{\text{PK}}^*_{\text{PKE}}, \hat{y}^*)$.
  5. $\mathcal{B}$ sets $\text{VK}_{\text{CVPRF}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\text{CVPRF}}[K^*\{(h^*, \ell^*)\}, K_{\text{PKE}}\{(h^*, \ell^*)\}, \widehat{\text{PK}}^*_{\text{PKE}}, \widehat{\text{CT}}^*_{\text{PKE}}, h^*, \ell^*]))$.
  6. $\mathcal{B}$ sets the challenge CVPRF value for $\mathcal{A}$ to be $y^* = r^*$.
  7. $\mathcal{B}$ provides $(\text{VK}_{\text{CVPRF}}, y^*)$ to $\mathcal{A}$.
- For $\eta = 1, \ldots, \hat{q}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K^{(\eta)}_1, \ldots, K^{(\eta)}_\lambda, K^{(\eta)}_{\text{SPS},A}, K^{(\eta)}_{\text{SPS},E} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. After that, it generates $(\text{PP}^{(\eta)}_{\text{ACC}}, w^{(\eta)}_0, \text{STORE}^{(\eta)}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}^{(\eta)}_{\text{ITR}}, v^{(\eta)}_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key

$$\text{SK}_{\text{CVPRF}}\{M^{(\eta)}\} = \left(\begin{array}{l} \text{HK}, \text{PP}^{(\eta)}_{\text{ACC}}, w^{(\eta)}_0, \text{STORE}^{(\eta)}_0, \text{PP}^{(\eta)}_{\text{ITR}}, v^{(\eta)}_0, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\eta)}_0, w^{(\eta)}_0, v^{(\eta)}_0, K^{(\eta)}_{\text{SPS},E}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\eta)}_{\text{ACC}}, \text{PP}^{(\eta)}_{\text{ITR}}, K^{(\eta)}_{\text{SPS},E}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\eta)}_{\text{SPS},A}, K^{(\eta)}_{\text{SPS},E}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\text{CVPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}^{(\eta)}_{\text{ACC}}, \text{PP}^{(\eta)}_{\text{ITR}}, K^*\{(h^*, \ell^*)\}, \\ \quad K_{\text{PKE}}\{(h^*, \ell^*)\}, K^{(\eta)}_1, \ldots, K^{(\eta)}_\lambda, K^{(\eta)}_{\text{SPS},A}, h^*, \ell^*]) \end{array}\right).$$

- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0, 1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its PPRF selective pseudorandomness experiment.

Note that the simulation of Hyb_5 by $\mathcal{B}$ is perfect. Also, if $\mathcal{A}$ wins in this simulated Hyb_5, then $\mathcal{B}$ wins in the PPRF selective pseudorandomness experiment against $\mathcal{F}$. □

Appendix C: Proof of Theorem 5.1

Theorem 5.1 (Security of the DCPRF Construction of Section 5.2). *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a secure positional accumulator as defined in Definition 2.4,* ITR *is a secure cryptographic iterator as per Definition 2.5,* SPS *is a secure splittable signature scheme according to Definition 2.6,* PRG *is a secure injective pseudorandom generator, and* PKE *is* CPA *secure, the* DCPRF *construction of Section 5.2 satisfies the correctness and selective pseudorandomness properties defined in Definition 5.1.*

Proof.

► **Correctness under Constraining/Delegation**: The correctness of the DCPRF construction of Section 5.2 follows readily from its construction and the correctness of the underlying cryptographic building blocks.

► **Selective Pseudorandomness**: Here also we devise a sequence of hybrid experiments in order to argue selective pseudorandomness of the DCPRF construction of Section 5.2. As earlier, we are working in the selective model and hence the challenger $\mathcal{B}$ knows the challenge input $x^* = x_0^* \dots x_{\ell^*-1}^* \in \mathcal{X}_{\text{DCPRF}}$ with $|x^*| = \ell^*$ and the SSB hash value $h^* = \mathcal{H}_{\text{HK}}(x^*)$ before receiving any key query from the adversary $\mathcal{A}$. Let $\hat{q}_{\text{CONST}}$ and $\hat{q}_{\text{DEL}}$ be respectively the total number of constrained and delegated key queries of the adversary $\mathcal{A}$. In view of Remark 5.1, here we consider no evaluation query and delegated key queries for only those pairs of TM's $(M^{(\theta)}, \widetilde{M}^{(\theta)}) \in \mathbb{M}_\lambda^2$ such that $[M^{(\theta)}(x^*) = 1] \ \wedge \ [\widetilde{M}^{(\theta)}(x^*) = 0]$. The sequence of hybrid experiments follows:

Sequence of Hybrid Experiments

Hyb_0: This experiment corresponds to the real selective pseudorandomness experiment described in Definition 5.1 for the DCPRF construction of Section 5.2. More formally, this experiment proceeds as follows:

- $\mathcal{A}$ submits a challenge input $x^* = x_0^* \dots x_{\ell^*-1}^* \in \mathcal{X}_{\text{DCPRF}}$ with $|x^*| = \ell^*$ to $\mathcal{B}$.
- $\mathcal{B}$ generates $\text{SK}_{\text{DCPRF}} = (K, \text{HK}) \xleftarrow{\$} \mathsf{DCPRF.Setup}(1^\lambda)$, as described in Section 5.2, and selects a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes $y^* = \mathsf{DCPRF}(\text{SK}_{\text{DCPRF}}, x^*) = \mathcal{F}(K, (h^* = \mathcal{H}_{\text{HK}}(x^*), \ell^*))$. Otherwise, $\mathcal{B}$ chooses $y^* \xleftarrow{\$} \mathcal{Y}_{\text{DCPRF}}$. $\mathcal{B}$ returns y^* to $\mathcal{A}$.
- For $\eta = 1, \dots, \hat{q}_{\text{CONST}}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} = \langle Q^{(\eta)}, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta^{(\eta)}, q_0^{(\eta)}, q_{\text{AC}}^{(\eta)}, q_{\text{REJ}}^{(\eta)} \rangle \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ generates $\text{SK}_{\text{DCPRF}}\{M^{(\eta)}\} = (K'^{(\eta)}, \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, v_0^{(\eta)}, \mathcal{P}_1^{(\eta)}, \mathcal{P}_2^{(\eta)}, \mathcal{P}_3^{(\eta)}, \mathcal{P}_{\text{DCPRF}}^{(\eta)}) \xleftarrow{\$} \mathsf{DCPRF.Constrain}(\text{SK}_{\text{DCPRF}}, M^{(\eta)})$ as described in Section 5.2. Here,
  - $\mathcal{P}_1^{(\eta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\text{SPS},E}^{(\eta)}])$,
  - $\mathcal{P}_2^{(\eta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}])$,
  - $\mathcal{P}_3^{(\eta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}])$,
  - $\mathcal{P}_{\text{DCPRF}}^{(\eta)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{DCPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K, K'^{(\eta)}, K_1^{(\eta)}, \dots, K_\lambda^{(\eta)}, K_{\text{SPS},A}^{(\eta)}])$.

  We assign the index η to all the components that are formed during the execution of $\mathsf{DCPRF.Constrain}(\text{SK}_{\text{DCPRF}}, M^{(\eta)})$.
- For $\theta = 1, \dots, \hat{q}_{\text{DEL}}$, in reply to the θ^{th} delegated key query of $\mathcal{A}$ corresponding to the TM pair $(M^{(\theta)} = \langle Q^{(\theta)}, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta^{(\theta)}, q_0^{(\theta)}, q_{\text{AC}}^{(\theta)}, q_{\text{REJ}}^{(\theta)} \rangle, \widetilde{M}^{(\theta)} = \langle \widetilde{Q}^{(\theta)}, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \widetilde{\delta}^{(\theta)}, \widetilde{q}_0^{(\theta)}, \widetilde{q}_{\text{AC}}^{(\theta)}, \widetilde{q}_{\text{REJ}}^{(\theta)} \rangle) \in \mathbb{M}_\lambda^2$ with $[M^{(\theta)}(x^*) = 1] \ \wedge \ [\widetilde{M}^{(\theta)}(x^*) = 0]$, $\mathcal{B}$ creates $\text{SK}_{\text{DCPRF}}\{M^{(\theta)} \ \wedge \ \widetilde{M}^{(\theta)}\} = (\widetilde{K'}^{(\theta)}, \text{HK}, \text{PP}_{\text{ACC}}^{(\theta)}, \widetilde{\text{PP}}_{\text{ACC}}^{(\theta)}, w_0^{(\theta)}, \widetilde{w}_0^{(\theta)}, \text{STORE}_0^{(\theta)}, \widetilde{\text{STORE}}_0^{(\theta)}, \text{PP}_{\text{ITR}}^{(\theta)}, \widetilde{\text{PP}}_{\text{ITR}}^{(\theta)}, v_0^{(\theta)}, \widetilde{v}_0^{(\theta)}, \mathcal{P}_1^{(\theta)}, \widetilde{\mathcal{P}}_1^{(\theta)}, \mathcal{P}_2^{(\theta)}, \widetilde{\mathcal{P}}_2^{(\theta)}, \mathcal{P}_3^{(\theta)}, \widetilde{\mathcal{P}}_3^{(\theta)}, \mathcal{P}_{\text{DCPRF}}^{(\theta)}, \widetilde{\mathcal{P}}_{\text{DCPRF}}^{(\theta)}) \xleftarrow{\$} \mathsf{DCPRF.Delegate}(\text{SK}_{\text{DCPRF}}\{M^{(\theta)}\}, \widetilde{M}^{(\theta)})$, as elaborated in Section 5.2, where either $\text{SK}_{\text{DCPRF}}\{M^{(\theta)}\} = \text{SK}_{\text{DCPRF}}\{M^{(\theta')}\}$, which has been generated while answering the θ'^{th} delegated key query of $\mathcal{A}$ for some $\theta' < \theta$, or $\text{SK}_{\text{DCPRF}}\{M^{(\theta)}\}$ is freshly generated in case it is not previously created. Here,
  - $\mathcal{P}_1^{(\theta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\theta)}, w_0^{(\theta)}, v_0^{(\theta)}, K_{\text{SPS},E}^{(\theta)}])$,
  - $\widetilde{\mathcal{P}}_1^{(\theta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[\widetilde{q}_0^{(\theta)}, \widetilde{w}_0^{(\theta)}, \widetilde{v}_0^{(\theta)}, \widetilde{K}_{\text{SPS},E}^{(\theta)}])$,
  - $\mathcal{P}_2^{(\theta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\theta)}, \text{PP}_{\text{ITR}}^{(\theta)}, K_{\text{SPS},E}^{(\theta)}])$,
  - $\widetilde{\mathcal{P}}_2^{(\theta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \widetilde{\text{PP}}_{\text{ACC}}^{(\theta)}, \widetilde{\text{PP}}_{\text{ITR}}^{(\theta)}, \widetilde{K}_{\text{SPS},E}^{(\theta)}])$,
  - $\mathcal{P}_3^{(\theta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\theta)}, K_{\text{SPS},E}^{(\theta)}])$,

– $\widetilde{\mathcal{P}}_3^{(\theta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[\widetilde{K}_{\mathrm{SPS},A}^{(\theta)}, \widetilde{K}_{\mathrm{SPS},E}^{(\theta)}])$,
– $\mathcal{P}_{\mathrm{DCPRF}}^{(\theta)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{DCPRF}}[M^{(\theta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\theta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\theta)}, K, K'^{(\theta)}, K_1^{(\theta)}, \ldots, K_\lambda^{(\theta)}, K_{\mathrm{SPS},A}^{(\theta)}])$,
– $\widetilde{\mathcal{P}}_{\mathrm{DCPRF}}^{(\theta)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{DCPRF}}[\widetilde{M}^{(\theta)}, T = 2^\lambda, \widetilde{\mathrm{PP}}_{\mathrm{ACC}}^{(\theta)}, \widetilde{\mathrm{PP}}_{\mathrm{ITR}}^{(\theta)}, K'^{(\theta)}, \widetilde{K}'^{(\theta)}, \widetilde{K}_1^{(\theta)}, \ldots, \widetilde{K}_\lambda^{(\theta)}, \widetilde{K}_{\mathrm{SPS},A}^{(\theta)}])$.

We assign the index θ to all the components that are generated while replying to the θ^{th} delegated key query. Ofcourse, if $\mathrm{SK}_{\mathrm{DCPRF}}\{M^{(\theta)}\}$ used for answering the query is $\mathrm{SK}_{\mathrm{DCPRF}}\{M^{(\theta')}\}$ for some $\theta' < \theta$, then all the delegated key components that are related to $\mathrm{SK}_{\mathrm{DCPRF}}\{M^{(\theta)}\}$ are the same as those included in $\mathrm{SK}_{\mathrm{DCPRF}}\{M^{(\theta')}\}$.

- $\mathcal{A}$ eventually outputs a guess bit $b' \in \{0,1\}$.

$\mathsf{Hyb}_{0,\nu}$ ($\nu = 1, \ldots, \hat{q}_{\mathrm{CONST}}$): This experiment is analogous to Hyb_0 with the exception that for $\eta \in [\hat{q}_{\mathrm{CONST}}]$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ returns the constrained key $\mathrm{SK}_{\mathrm{DCPRF}}\{M^{(\eta)}\} = (K'^{(\eta)}, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \mathcal{P}_1^{(\eta)}, \mathcal{P}_2^{(\eta)}, \mathcal{P}_3^{(\eta)}, \mathcal{P}_{\mathrm{DCPRF}}^{(\eta)})$, where

– $\mathcal{P}_1^{(\eta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}])$,
– $\mathcal{P}_2^{(\eta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}])$,
– $\mathcal{P}_3^{(\eta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}])$,
– $\mathcal{P}_{\mathrm{DCPRF}}^{(\eta)} = \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{DCPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K'^{(\eta)}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]})$

to $\mathcal{A}$, if $\underline{\eta \leq \nu}$, while $\mathcal{B}$ gives the constrained key $\mathrm{SK}_{\mathrm{DCPRF}}\{M^{(\eta)}\} = (K'^{(\eta)}, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \mathcal{P}_1^{(\eta)}, \mathcal{P}_2^{(\eta)}, \mathcal{P}_3^{(\eta)}, \mathcal{P}_{\mathrm{DCPRF}}^{(\eta)})$, where

– $\mathcal{P}_1^{(\eta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}])$,
– $\mathcal{P}_2^{(\eta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}])$,
– $\mathcal{P}_3^{(\eta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}])$,
– $\mathcal{P}_{\mathrm{DCPRF}}^{(\eta)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{DCPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K'^{(\eta)}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}])$

to $\mathcal{A}$, if $\underline{\eta > \nu}$. The program $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{DCPRF}}$ is an alteration of the program $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{DCPRF}}$ (Fig. 5.1) and is depicted in Fig. C.1. Observe that $\mathsf{Hyb}_{0,0}$ coincides Hyb_0.

Hyb_1: This experiment coincides with $\mathsf{Hyb}_{0,\hat{q}_{\mathrm{CONST}}}$.

$\mathsf{Hyb}_{1,\omega}$ ($\omega = 1, \ldots, \hat{q}_{\mathrm{DEL}}$): This experiment is similar to Hyb_1 except that for $\theta \in [\hat{q}_{\mathrm{DEL}}]$, in reply to the θ^{th} delegated key query of $\mathcal{A}$ corresponding to TM pair $(M^{(\theta)}, \widetilde{M}^{(\theta)}) \in \mathbb{M}_\lambda^2$ with $[M^{(\theta)}(x^*) = 1] \wedge [\widetilde{M}^{(\theta)}(x^*) = 0]$, $\mathcal{B}$ returns the delegated key $\mathrm{SK}_{\mathrm{DCPRF}}\{M^{(\theta)} \wedge \widetilde{M}^{(\theta)}\} = (\widetilde{K}'^{(\theta)}, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\theta)}, \widetilde{\mathrm{PP}}_{\mathrm{ACC}}^{(\theta)}, w_0^{(\theta)}, \widetilde{w}_0^{(\theta)}, \mathrm{STORE}_0^{(\theta)}, \widetilde{\mathrm{STORE}}_0^{(\theta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\theta)}, \widetilde{\mathrm{PP}}_{\mathrm{ITR}}^{(\theta)}, v_0^{(\theta)}, \widetilde{v}_0^{(\theta)}, \mathcal{P}_1^{(\theta)}, \widetilde{\mathcal{P}}_1^{(\theta)}, \mathcal{P}_2^{(\theta)}, \widetilde{\mathcal{P}}_2^{(\theta)}, \mathcal{P}_3^{(\theta)}, \widetilde{\mathcal{P}}_3^{(\theta)}, \mathcal{P}_{\mathrm{DCPRF}}^{(\theta)}, \widetilde{\mathcal{P}}_{\mathrm{DCPRF}}^{(\theta)})$, where

– $\mathcal{P}_1^{(\theta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\theta)}, w_0^{(\theta)}, v_0^{(\theta)}, K_{\mathrm{SPS},E}^{(\theta)}])$,
– $\widetilde{\mathcal{P}}_1^{(\theta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[\widetilde{q}_0^{(\theta)}, \widetilde{w}_0^{(\theta)}, \widetilde{v}_0^{(\theta)}, \widetilde{K}_{\mathrm{SPS},E}^{(\theta)}])$,
– $\mathcal{P}_2^{(\theta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\theta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\theta)}, K_{\mathrm{SPS},E}^{(\theta)}])$,
– $\widetilde{\mathcal{P}}_2^{(\theta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \widetilde{\mathrm{PP}}_{\mathrm{ACC}}^{(\theta)}, \widetilde{\mathrm{PP}}_{\mathrm{ITR}}^{(\theta)}, \widetilde{K}_{\mathrm{SPS},E}^{(\theta)}])$,

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K', K_1, \ldots, K_\lambda, K_{\text{SPS},A}$, SSB hash value of challenge input h^*, Length of the challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Encryption of DCPRF value CT_{PKE}, or Header Position (POS_{OUT}, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   (b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \ \wedge \ [(h, \ell_{\text{INP}}) \neq (h^*, \ell^*)]$, perform the following steps:
   (I) Compute $r_{\text{PKE},1} \| r_{\text{PKE},2} = \mathcal{F}(K', (h, \ell_{\text{INP}}))$, $(\text{PK}_{\text{PKE}}, \text{SK}_{\text{PKE}}) = \mathsf{PKE.Setup}(1^\lambda;\ r_{\text{PKE},1})$.
   (II) Output $\text{CT}_{\text{PKE}} = \mathsf{PKE.Encrypt}(\text{PK}_{\text{PKE}}, \mathcal{F}(K, (h, \ell_{\text{INP}}));\ r_{\text{PKE},2})$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, output $\perp$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. C.1. $\mathsf{Constrained\text{-}Key.Prog}'_{\text{DCPRF}}$

- $\mathcal{P}_3^{(\theta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\theta)}, K_{\text{SPS},E}^{(\theta)}])$,
- $\widetilde{\mathcal{P}}_3^{(\theta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[\widetilde{K}_{\text{SPS},A}^{(\theta)}, \widetilde{K}_{\text{SPS},E}^{(\theta)}])$,
- $\mathcal{P}_{\text{DCPRF}}^{(\theta)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{DCPRF}}[M^{(\theta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\theta)}, \text{PP}_{\text{ITR}}^{(\theta)}, K, K'^{(\theta)}, K_1^{(\theta)}, \ldots, K_\lambda^{(\theta)}, K_{\text{SPS},A}^{(\theta)}])$,
- $\widetilde{\mathcal{P}}_{\text{DCPRF}}^{(\theta)} = \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}'_{\text{DCPRF}}[\widetilde{M}^{(\theta)}, T = 2^\lambda, \widetilde{\text{PP}}_{\text{ACC}}^{(\theta)}, \widetilde{\text{PP}}_{\text{ITR}}^{(\theta)}, K'^{(\theta)}, \widetilde{K}'^{(\theta)}, \widetilde{K}_1^{(\theta)}, \ldots, \widetilde{K}_\lambda^{(\theta)}, \widetilde{K}_{\text{SPS},A}^{(\theta)}, h^*, \ell^*]})$

to $\mathcal{A}$, if $\underline{\theta \leq \omega}$, while $\mathcal{B}$ gives the delegated key $\text{SK}_{\text{DCPRF}}\{M^{(\theta)} \wedge \widetilde{M}^{(\theta)}\} = (\widetilde{K}'^{(\theta)}, \text{HK}, \text{PP}_{\text{ACC}}^{(\theta)}, \widetilde{\text{PP}}_{\text{ACC}}^{(\theta)}, w_0^{(\theta)}, \widetilde{w}_0^{(\theta)}, \text{STORE}_0^{(\theta)}, \widetilde{\text{STORE}}_0^{(\theta)}, \text{PP}_{\text{ITR}}^{(\theta)}, \widetilde{\text{PP}}_{\text{ITR}}^{(\theta)}, v_0^{(\theta)}, \widetilde{v}_0^{(\theta)}, \mathcal{P}_1^{(\theta)}, \widetilde{\mathcal{P}}_1^{(\theta)}, \mathcal{P}_2^{(\theta)}, \widetilde{\mathcal{P}}_2^{(\theta)}, \mathcal{P}_3^{(\theta)}, \widetilde{\mathcal{P}}_3^{(\theta)}, \mathcal{P}_{\text{DCPRF}}^{(\theta)}, \widetilde{\mathcal{P}}_{\text{DCPRF}}^{(\theta)})$, where

- $\mathcal{P}_1^{(\theta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\theta)}, w_0^{(\theta)}, v_0^{(\theta)}, K_{\text{SPS},E}^{(\theta)}])$,
- $\widetilde{\mathcal{P}}_1^{(\theta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[\widetilde{q}_0^{(\theta)}, \widetilde{w}_0^{(\theta)}, \widetilde{v}_0^{(\theta)}, \widetilde{K}_{\text{SPS},E}^{(\theta)}])$,
- $\mathcal{P}_2^{(\theta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\theta)}, \text{PP}_{\text{ITR}}^{(\theta)}, K_{\text{SPS},E}^{(\theta)}])$,
- $\widetilde{\mathcal{P}}_2^{(\theta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \widetilde{\text{PP}}_{\text{ACC}}^{(\theta)}, \widetilde{\text{PP}}_{\text{ITR}}^{(\theta)}, \widetilde{K}_{\text{SPS},E}^{(\theta)}])$,
- $\mathcal{P}_3^{(\theta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\theta)}, K_{\text{SPS},E}^{(\theta)}])$,
- $\widetilde{\mathcal{P}}_3^{(\theta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[\widetilde{K}_{\text{SPS},A}^{(\theta)}, \widetilde{K}_{\text{SPS},E}^{(\theta)}])$,
- $\mathcal{P}_{\text{DCPRF}}^{(\theta)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{DCPRF}}[M^{(\theta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\theta)}, \text{PP}_{\text{ITR}}^{(\theta)}, K, K'^{(\theta)}, K_1^{(\theta)}, \ldots, K_\lambda^{(\theta)}, K_{\text{SPS},A}^{(\theta)}])$,
- $\widetilde{\mathcal{P}}_{\text{DCPRF}}^{(\theta)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{DCPRF}}[\widetilde{M}^{(\theta)}, T = 2^\lambda, \widetilde{\text{PP}}_{\text{ACC}}^{(\theta)}, \widetilde{\text{PP}}_{\text{ITR}}^{(\theta)}, K'^{(\theta)}, \widetilde{K}'^{(\theta)}, \widetilde{K}_1^{(\theta)}, \ldots, \widetilde{K}_\lambda^{(\theta)}, \widetilde{K}_{\text{SPS},A}^{(\theta)}])$

to $\mathcal{A}$, if $\theta > \omega$. Observe that $\mathsf{Hyb}_{1,0}$ coincides with Hyb_1.

Hyb_2: This experiment coincides with $\mathsf{Hyb}_{1,\hat{q}_{\mathrm{DEL}}}$.

Hyb_3: This experiment is identical to Hyb_2 except that for $\theta = 1, \ldots, \hat{q}_{\mathrm{DEL}}$, in response to the θ^{th} delegated key query of $\mathcal{A}$ corresponding to TM pair $(M^{(\theta)}, \widetilde{M}^{(\theta)}) \in \mathbb{M}_\lambda^2$ with $[M^{(\theta)}(x^*) = 1] \wedge [\widetilde{M}^{(\theta)}(x^*) = 0]$, $\mathcal{B}$ returns the delegated key $\mathrm{SK}_{\mathrm{DCPRF}}\{M^{(\theta)} \wedge \widetilde{M}^{(\theta)}\} = (\widetilde{K}'^{(\theta)}, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\theta)}, \widetilde{\mathrm{PP}}_{\mathrm{ACC}}^{(\theta)}, w_0^{(\theta)}, \widetilde{w}_0^{(\theta)}, \mathrm{STORE}_0^{(\theta)}, \widetilde{\mathrm{STORE}}_0^{(\theta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\theta)}, \widetilde{\mathrm{PP}}_{\mathrm{ITR}}^{(\theta)}, v_0^{(\theta)}, \widetilde{v}_0^{(\theta)}, \mathcal{P}_1^{(\theta)}, \widetilde{\mathcal{P}}_1^{(\theta)}, \mathcal{P}_2^{(\theta)}, \widetilde{\mathcal{P}}_2^{(\theta)}, \mathcal{P}_3^{(\theta)}, \widetilde{\mathcal{P}}_3^{(\theta)}, \mathcal{P}_{\mathrm{DCPRF}}^{(\theta)}, \widetilde{\mathcal{P}}_{\mathrm{DCPRF}}^{(\theta)})$ to $\mathcal{A}$, where

- $\mathcal{P}_1^{(\theta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\theta)}, w_0^{(\theta)}, v_0^{(\theta)}, K_{\mathrm{SPS},E}^{(\theta)}])$,
- $\widetilde{\mathcal{P}}_1^{(\theta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[\widetilde{q}_0^{(\theta)}, \widetilde{w}_0^{(\theta)}, \widetilde{v}_0^{(\theta)}, \widetilde{K}_{\mathrm{SPS},E}^{(\theta)}])$,
- $\mathcal{P}_2^{(\theta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\mathrm{SSB\text{-}BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\theta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\theta)}, K_{\mathrm{SPS},E}^{(\theta)}])$,
- $\widetilde{\mathcal{P}}_2^{(\theta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\mathrm{SSB\text{-}BLK}} = 2^\lambda, \mathrm{HK}, \widetilde{\mathrm{PP}}_{\mathrm{ACC}}^{(\theta)}, \widetilde{\mathrm{PP}}_{\mathrm{ITR}}^{(\theta)}, \widetilde{K}_{\mathrm{SPS},E}^{(\theta)}])$,
- $\mathcal{P}_3^{(\theta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\theta)}, K_{\mathrm{SPS},E}^{(\theta)}])$,
- $\widetilde{\mathcal{P}}_3^{(\theta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[\widetilde{K}_{\mathrm{SPS},A}^{(\theta)}, \widetilde{K}_{\mathrm{SPS},E}^{(\theta)}])$,
- $\mathcal{P}_{\mathrm{DCPRF}}^{(\theta)} = \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}''_{\mathrm{DCPRF}}[M^{(\theta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\theta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\theta)}, K, K'^{(\theta)}\{(h^*, \ell^*)\}, K_1^{(\theta)}, \ldots, K_\lambda^{(\theta)}, K_{\mathrm{SPS},A}^{(\theta)}, \mathrm{CT}_{\mathrm{PKE}}^{*(\theta)}, h^*, \ell^*]})$,
- $\widetilde{\mathcal{P}}_{\mathrm{DCPRF}}^{(\theta)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{DCPRF}}[\widetilde{M}^{(\theta)}, T = 2^\lambda, \widetilde{\mathrm{PP}}_{\mathrm{ACC}}^{(\theta)}, \widetilde{\mathrm{PP}}_{\mathrm{ITR}}^{(\theta)}, K'^{(\theta)}\{(h^*, \ell^*)\}, \widetilde{K}'^{(\theta)}, \widetilde{K}_1^{(\theta)}, \ldots, \widetilde{K}_\lambda^{(\theta)}, \widetilde{K}_{\mathrm{SPS},A}^{(\theta)}, h^*, \ell^*])$

such that $\underline{K'^{(\theta)}\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K'^{(\theta)}, (h^*, \ell^*)), r_{\mathrm{PKE},1}^{*(\theta)} \| r_{\mathrm{PKE},2}^{*(\theta)} = \mathcal{F}(K'^{(\theta)}, (h^*, \ell^*)), (\mathrm{PK}_{\mathrm{PKE}}^{*(\theta)}, \mathrm{SK}_{\mathrm{PKE}}^{*(\theta)}) = \mathsf{PKE.Setup}(1^\lambda;\ r_{\mathrm{PKE},1}^{*(\theta)}), \mathrm{CT}_{\mathrm{PKE}}^{*(\theta)} = \mathsf{PKE.Encrypt}(\mathrm{PK}_{\mathrm{PKE}}^{*(\theta)}, \mathcal{F}(K, (h^*, \ell^*));\ r_{\mathrm{PKE},2}^{*(\theta)})}$, and the program $\underline{\mathsf{Constrained\text{-}Key.Prog}''_{\mathrm{DCPRF}}}$ is a modification of the program $\underline{\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{DCPRF}}}$ (Fig. C.1) and is shown in Fig. C.2. As in the previous hybrid experiments, here also once the components pertaining to some parent TM $M^{(\theta)} \in \mathbb{M}_\lambda$ is generated while answering the θ^{th} delegated key query $(M^{(\theta)}, \widetilde{M}^{(\theta)})$ those are reused in all the subsequent delegated key queries with the same parent TM $M^{(\theta)}$.

$\mathsf{Hyb}_{3,\omega}$ ($\omega = 1, \ldots, \hat{q}_{\mathrm{DEL}}$): This experiment is the same as $\mathsf{Hyb}_{3,(\omega-1)'}$ with the only exception that while constructing the ω^{th} delegated key corresponding to TM pair $(M^{(\omega)}, \widetilde{M}^{(\omega)}) \in \mathbb{M}_\lambda^2$ with $[M^{(\omega)}(x^*) = 1] \wedge [\widetilde{M}^{(\omega)}(x^*) = 0]$, if $M^{(\omega)}$ has not yet appeared in any previous delegated key query of $\mathcal{A}$, then $\mathcal{B}$ selects $\underline{r_{\mathrm{PKE},1}^{*(\omega)} \| r_{\mathrm{PKE},2}^{*(\omega)} \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}}$, i.e., $\mathcal{B}$ creates $\underline{(\mathrm{PK}_{\mathrm{PKE}}^{*(\omega)}, \mathrm{SK}_{\mathrm{PKE}}^{*(\omega)}) \xleftarrow{\$} \mathsf{PKE.Setup}(1^\lambda)}$ and forms $\underline{\mathrm{CT}_{\mathrm{PKE}}^{*(\omega)} \xleftarrow{\$} \mathsf{PKE.Encrypt}(\mathrm{PK}_{\mathrm{PKE}}^{*(\omega)}, \mathcal{F}(K, (h^*, \ell^*)))}$; while if $M^{(\omega)}$ has already appeared in previous delegated key query of $\mathcal{A}$, then $\mathcal{B}$ utilizes the already generated values corresponding to $M^{(\omega)}$ like in the earlier hybrid experiments.

$\mathsf{Hyb}_{3,\omega'}$ ($\omega = 1, \ldots, \hat{q}_{\mathrm{DEL}}$): This experiment is analogous to $\mathsf{Hyb}_{3,\omega}$ with the only exception that while constructing the ω^{th} delegated key queried by $\mathcal{A}$ corresponding to TM pair $(M^{(\omega)}, \widetilde{M}^{(\omega)}) \in \mathbb{M}_\lambda^2$ with $[M^{(\omega)}(x^*) = 1] \wedge [\widetilde{M}^{(\omega)}(x^*) = 0]$, if $M^{(\omega)}$ has not yet appeared in any previous delegated key query of $\mathcal{A}$, then $\mathcal{B}$ forms $\underline{\mathrm{CT}_{\mathrm{PKE}}^{*(\omega)} \xleftarrow{\$} \mathsf{PKE.Encrypt}(\mathrm{PK}_{\mathrm{PKE}}^{*(\omega)}, \hat{y}^{*(\omega)})}$, where $\underline{\hat{y}^{*(\omega)} \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}}$, whereas, if $M^{(\omega)}$ has already appeared in some earlier delegated key query of $\mathcal{A}$, then $\mathcal{B}$ uses the already created values corresponding to $M^{(\omega)}$ just as in the previous hybrid experiments. Observe that $\mathsf{Hyb}_{3,0'}$ coincides with Hyb_3.

Hyb_4: This experiment is similar to $\mathsf{Hyb}_{3,\hat{q}_{\mathrm{DEL}}}$ other than the following exceptions:

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF key K, Puncture PPRF key $K'\{(h^*, \ell^*)\}$, PPRF keys $K_1, \ldots, K_\lambda, K_{\text{SPS},A}$, PKE ciphertext CT^*_{PKE}, SSB hash value of challenge input h^*, Length of the challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: Encryption of DCPRF value CT_{PKE}, or Header Position (POS_{OUT}, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   (b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[(\text{ST}_{\text{OUT}} = q_{\text{AC}}) \ \wedge \ [(h, \ell_{\text{INP}}) = (h^*, \ell^*)]$, output CT^*_{PKE}.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, perform the following steps:
   (I) Compute $r_{\text{PKE},1} \| r_{\text{PKE},2} = \mathcal{F}(K', (h, \ell_{\text{INP}}))$, $(\text{PK}_{\text{PKE}}, \text{SK}_{\text{PKE}} = \mathsf{PKE.Setup}(1^\lambda;\ r_{\text{PKE},1})$.
   (II) Output $\text{CT}_{\text{PKE}} = \mathsf{PKE.Encrypt}(\text{PK}_{\text{PKE}}, \mathcal{F}(K, (h, \ell_{\text{INP}}));\ r_{\text{PKE},2})$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$.
   Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$.
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. C.2. $\mathsf{Constrained\text{-}Key.Prog}''_{\text{DCPRF}}$

(I) For $\eta = 1, \ldots, \hat{q}_{\text{CONST}}$, in response to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ returns the constrained key $\text{SK}_{\text{DCPRF}}\{M^{(\eta)}\} = (K'^{(\eta)}, \text{HK}, \text{PP}^{(\eta)}_{\text{ACC}}, w^{(\eta)}_0, \text{STORE}^{(\eta)}_0, \text{PP}^{(\eta)}_{\text{ITR}}, v^{(\eta)}_0, \mathcal{P}^{(\eta)}_1, \mathcal{P}^{(\eta)}_2, \mathcal{P}^{(\eta)}_3, \mathcal{P}^{(\eta)}_{\text{DCPRF}})$ to $\mathcal{A}$, where
- $\mathcal{P}^{(\eta)}_1 = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\eta)}_0, w^{(\eta)}_0, v^{(\eta)}_0, K^{(\eta)}_{\text{SPS},E}])$,
- $\mathcal{P}^{(\eta)}_2 = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\eta)}_{\text{ACC}}, \text{PP}^{(\eta)}_{\text{ITR}}, K^{(\eta)}_{\text{SPS},E}])$,
- $\mathcal{P}^{(\eta)}_3 = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\eta)}_{\text{SPS},A}, K^{(\eta)}_{\text{SPS},E}])$,
- $\mathcal{P}^{(\eta)}_{\text{DCPRF}} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\text{DCPRF}}[M^{(\eta)}, T = 2^\lambda, \text{PP}^{(\eta)}_{\text{ACC}}, \text{PP}^{(\eta)}_{\text{ITR}}, K\{(h^*, \ell^*)\}, K'^{(\eta)}, K^{(\eta)}_1, \ldots, K^{(\eta)}_\lambda, K^{(\eta)}_{\text{SPS},A}, h^*, \ell^*])$.

(II) For $\theta = 1, \ldots, \hat{q}_{\text{DEL}}$, in response to the θ^{th} delegated key query of $\mathcal{A}$ corresponding to TM pair $(M^{(\theta)}, \widetilde{M}^{(\theta)}) \in \mathbb{M}^2_\lambda$ with $[M^{(\theta)}(x^*) = 1] \ \wedge \ [\widetilde{M}^{(\theta)}(x^*) = 0]$, $\mathcal{B}$ returns the delegated key $\text{SK}_{\text{DCPRF}}\{M^{(\theta)} \ \wedge \ \widetilde{M}^{(\theta)}\} = (\widetilde{K'}^{(\theta)}, \text{HK}, \text{PP}^{(\theta)}_{\text{ACC}}, \widetilde{\text{PP}}^{(\theta)}_{\text{ACC}}, w^{(\theta)}_0, \widetilde{w}^{(\theta)}_0, \text{STORE}^{(\theta)}_0, \widetilde{\text{STORE}}^{(\theta)}_0, \text{PP}^{(\theta)}_{\text{ITR}}, \widetilde{\text{PP}}^{(\theta)}_{\text{ITR}}, v^{(\theta)}_0, \widetilde{v}^{(\theta)}_0, \mathcal{P}^{(\theta)}_1, \widetilde{\mathcal{P}}^{(\theta)}_1, \mathcal{P}^{(\theta)}_2, \widetilde{\mathcal{P}}^{(\theta)}_2, \mathcal{P}^{(\theta)}_3, \widetilde{\mathcal{P}}^{(\theta)}_3, \mathcal{P}^{(\theta)}_{\text{DCPRF}}, \widetilde{\mathcal{P}}^{(\theta)}_{\text{DCPRF}})$ to $\mathcal{A}$, where
- $\mathcal{P}^{(\theta)}_1 = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\theta)}_0, w^{(\theta)}_0, v^{(\theta)}_0, K^{(\theta)}_{\text{SPS},E}])$,
- $\widetilde{\mathcal{P}}^{(\theta)}_1 = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[\widetilde{q}^{(\theta)}_0, \widetilde{w}^{(\theta)}_0, \widetilde{v}^{(\theta)}_0, \widetilde{K}^{(\theta)}_{\text{SPS},E}])$,
- $\mathcal{P}^{(\theta)}_2 = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\theta)}_{\text{ACC}}, \text{PP}^{(\theta)}_{\text{ITR}}, K^{(\theta)}_{\text{SPS},E}])$,
- $\widetilde{\mathcal{P}}^{(\theta)}_2 = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \widetilde{\text{PP}}^{(\theta)}_{\text{ACC}}, \widetilde{\text{PP}}^{(\theta)}_{\text{ITR}}, \widetilde{K}^{(\theta)}_{\text{SPS},E}])$,
- $\mathcal{P}^{(\theta)}_3 = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\theta)}_{\text{SPS},A}, K^{(\theta)}_{\text{SPS},E}])$,
- $\widetilde{\mathcal{P}}^{(\theta)}_3 = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[\widetilde{K}^{(\theta)}_{\text{SPS},A}, \widetilde{K}^{(\theta)}_{\text{SPS},E}])$,

– $\mathcal{P}^{(\theta)}_{\text{DCPRF}} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}''_{\text{DCPRF}}[M^{(\theta)}, T = 2^\lambda, \text{PP}^{(\theta)}_{\text{ACC}}, \text{PP}^{(\theta)}_{\text{ITR}}, \underline{K\{(h^*, \ell^*)\}}, K'^{(\theta)}\{(h^*, \ell^*)\}, K^{(\theta)}_1, \ldots, K^{(\theta)}_\lambda, K^{(\theta)}_{\text{SPS},A}, \text{CT}^{*(\theta)}_{\text{PKE}}, h^*, \ell^*])$,
– $\widetilde{\mathcal{P}}^{(\theta)}_{\text{DCPRF}} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\text{DCPRF}}[\widetilde{M}^{(\theta)}, T = 2^\lambda, \widetilde{\text{PP}}^{(\theta)}_{\text{ACC}}, \widetilde{\text{PP}}^{(\theta)}_{\text{ITR}}, K'^{(\theta)}\{(h^*, \ell^*)\}, \widetilde{K'^{(\theta)}}, \widetilde{K}^{(\theta)}_1, \ldots, \widetilde{K}^{(\theta)}_\lambda, \widetilde{K}^{(\theta)}_{\text{SPS},A}, h^*, \ell^*])$.

Here, $\underline{K\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K, (h^*, \ell^*))}$.

Analysis

Let $\mathsf{Adv}^{(0)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(0,\nu)}_{\mathcal{A}}(\lambda)$ $(\nu = 1, \ldots, \hat{q}_{\text{CONST}}), \mathsf{Adv}^{(1)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(1,\omega)}_{\mathcal{A}}(\lambda)$ $(\omega = 1, \ldots, \hat{q}_{\text{DEL}}), \mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda)$, $\mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(3,\omega)}_{\mathcal{A}}(\lambda)$ $(\omega = 1, \ldots, \hat{q}_{\text{DEL}}), \mathsf{Adv}^{(3,\omega')}_{\mathcal{A}}(\lambda)$ $(\omega = 1, \ldots, \hat{q}_{\text{DEL}})$, and $\mathsf{Adv}^{(4)}_{\mathcal{A}}(\lambda)$ represent respectively the advantage of the adversary $\mathcal{A}$, i.e., the absolute difference between $1/2$ and $\mathcal{A}$'s probability of correctly guessing the random bit selected by the challenger $\mathcal{B}$, in the hybrid experiment Hyb_Υ with Υ as indicated in the superscript of the advantage notation. From the description of the hybrid experiments it follows that $\mathsf{Adv}^{\text{DCPRF,SEL-PR}}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,0)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(1)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(0,\hat{q}_{\text{CONST}})}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(1,0)}_{\mathcal{A}}(\lambda), \mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(1,\hat{q}_{\text{DEL}})}_{\mathcal{A}}(\lambda)$, and $\mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda) \equiv \mathsf{Adv}^{(3,0')}_{\mathcal{A}}(\lambda)$. Therefore, we have

$$
\begin{aligned}
&\mathsf{Adv}^{\text{DCPRF,SEL-PR}}_{\mathcal{A}}(\lambda) \\
&\leq \sum_{\nu=1}^{\hat{q}_{\text{CONST}}} |\mathsf{Adv}^{(0,\nu-1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu)}_{\mathcal{A}}(\lambda)| + \sum_{\omega=1}^{\hat{q}_{\text{DEL}}} |\mathsf{Adv}^{(1,\omega-1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(1,\omega)}_{\mathcal{A}}(\lambda)| + \\
&\quad |\mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda)| + \sum_{\omega=1}^{\hat{q}_{\text{DEL}}} |\mathsf{Adv}^{(3,(\omega-1)')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(3,\omega)}_{\mathcal{A}}(\lambda)| + \\
&\quad \sum_{\omega=1}^{\hat{q}_{\text{DEL}}} |\mathsf{Adv}^{(3,\omega)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(3,\omega')}_{\mathcal{A}}(\lambda)| + |\mathsf{Adv}^{(3,\hat{q}'_{\text{DEL}})}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(4)}_{\mathcal{A}}(\lambda)| + \mathsf{Adv}^{(4)}_{\mathcal{A}}(\lambda).
\end{aligned}
\tag{C.1}
$$

Lemmas C.1–C.7 will show that the RHS of Eq. (C.1) is negligible and hence Theorem 5.1 follows. □

C.1 Lemmas for the Proof of Theorem 5.1

Lemma C.1. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a secure positional accumulator as defined in Definition 2.4,* ITR *is a secure cryptographic iterator according to Definition 2.5,* SPS *is a secure splittable signature scheme as per Definition 2.6, and* PRG *is a secure injective pseudorandom generator, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(0,\nu-1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(0,\nu)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Lemma C.1 proceeds in exactly the same path as that of Lemma A.3. We omit the details to avoid repetition. □

Lemma C.2. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly*, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2,* SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a secure positional accumulator as defined in Definition 2.4,* ITR *is a secure cryptographic iterator according to Definition 2.5,* SPS *is a secure splittable signature scheme as per Definition 2.6, and* PRG *is a secure pseudorandom generator, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(1,\omega-1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(1,\omega)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The proof of Lemma C.2 again takes an identical path as that taken by the proof of Lemma A.3. The details are omitted to avoid repetition. □

Lemma C.3. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$ *and $\mathcal{F}$ satisfies the correctness under puncturing property described in Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The difference between Hyb_2 and Hyb_3 is the following: For $\theta \in [\hat{q}_{\mathrm{DEL}}]$, within the θ^{th} delegated key returned to $\mathcal{A}$, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_0^{(\theta)})$ and $\mathcal{IO}(\widetilde{P}_0^{(\theta)})$ in Hyb_2, whereas, in Hyb_3, $\mathcal{B}$ includes the programs $\mathcal{IO}(P_1^{(\theta)})$ and $\mathcal{IO}(\widetilde{P}_1^{(\theta)})$ instead, where

- $P_0^{(\theta)} = \mathsf{Constrained\text{-}Key.Prog}_{\mathrm{DCPRF}}[M^{(\theta)}, T = 2^\lambda, \mathrm{PP}^{(\theta)}_{\mathrm{ACC}}, \mathrm{PP}^{(\theta)}_{\mathrm{ITR}}, K, K'^{(\theta)}, K^{(\theta)}_1, \ldots, K^{(\theta)}_\lambda, K^{(\theta)}_{\mathrm{SPS},A}]$ (Fig. 5.1),
- $\widetilde{P}_0^{(\theta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{DCPRF}}[\widetilde{M}^{(\theta)}, T = 2^\lambda, \widetilde{\mathrm{PP}}^{(\theta)}_{\mathrm{ACC}}, \widetilde{\mathrm{PP}}^{(\theta)}_{\mathrm{ITR}}, K'^{(\theta)}, \widetilde{K}'^{(\theta)}, \widetilde{K}^{(\theta)}_1, \ldots, \widetilde{K}^{(\theta)}_\lambda, \widetilde{K}^{(\theta)}_{\mathrm{SPS},A}, h^*, \ell^*]$ (Fig. C.1),
- $P_1^{(\theta)} = \mathsf{Constrained\text{-}Key.Prog}''_{\mathrm{DCPRF}}[M^{(\theta)}, T = 2^\lambda, \mathrm{PP}^{(\theta)}_{\mathrm{ACC}}, \mathrm{PP}^{(\theta)}_{\mathrm{ITR}}, K, K'^{(\theta)}\{(h^*, \ell^*)\}, K^{(\theta)}_1, \ldots, K^{(\theta)}_\lambda, K^{(\theta)}_{\mathrm{SPS},A}, \mathrm{CT}^{*(\theta)}_{\mathrm{PKE}}, h^*, \ell^*]$ (Fig. C.2),
- $\widetilde{P}_1^{(\theta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{DCPRF}}[\widetilde{M}^{(\theta)}, T = 2^\lambda, \widetilde{\mathrm{PP}}^{(\theta)}_{\mathrm{ACC}}, \widetilde{\mathrm{PP}}^{(\theta)}_{\mathrm{ITR}}, K'^{(\theta)}\{(h^*, \ell^*)\}, \widetilde{K}'^{(\theta)}, \widetilde{K}^{(\theta)}_1, \ldots, \widetilde{K}^{(\theta)}_\lambda, \widetilde{K}^{(\theta)}_{\mathrm{SPS},A}, h^*, \ell^*]$ (Fig. C.1).

We will argue that the programs $P_0^{(\theta)}$ and $P_1^{(\theta)}$, as well as, the programs $\widetilde{P}_0^{(\theta)}$ and $\widetilde{P}_1^{(\theta)}$ are functionally equivalent, so that, by the security of $\mathcal{IO}$, Lemma C.3 follows. First, note that $P_0^{(\theta)}$ and $P_1^{(\theta)}$ are functionally identical since by the correctness under puncturing property of the PPRF $\mathcal{F}$, the two programs clearly have identical outputs on inputs corresponding to $(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)$, while for inputs corresponding to (h^*, ℓ^*), the hardwired ciphertext outputted by $P_1^{(\theta)}$ is exactly the one computed by $P_0^{(\theta)}$ in this case. The programs $\widetilde{P}_0^{(\theta)}$ and $\widetilde{P}_1^{(\theta)}$ are also functionally equivalent as both programs output $\perp$ for inputs corresponding to (h^*, ℓ^*) and by the correctness under puncturing property of the PPRF $\mathcal{F}$ the programs clearly have the same outputs for inputs corresponding to $(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)$. Ofcourse, to reach the conclusion, we would have to move through a sequence of hybrid experiments where in each hybrid experiment we change the programs one at a time. □

Lemma C.4. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(3,(\omega-1)')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(3,\omega)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. We will only focus on the case where the parent TM $M^{(\omega)} \in \mathbb{M}_\lambda$ of the ω^{th} delegated key query $(M^{(\omega)}, \widetilde{M}^{(\omega)}) \in \mathbb{M}^2_\lambda$ made by $\mathcal{A}$ is not one that has appeared in some previous delegated key query of $\mathcal{A}$. This is because in the other case by the description of the hybrid experiments it follows that $|\mathsf{Adv}^{(3,(\omega-1)')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(3,\omega)}_{\mathcal{A}}(\lambda)| = 0$.

Suppose there is a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}^{(3,(\omega-1)')}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(3,\omega)}_{\mathcal{A}}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x^*_0 \ldots x^*_{\ell^*-1} \in \mathcal{X}_{\mathrm{DCPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.
  2. Next, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.

3. Then $\mathcal{B}$ chooses a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, then $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. Otherwise, it picks $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
4. $\mathcal{B}$ returns the challenge DCPRF value y^* to $\mathcal{A}$.

- For $\eta \in [\hat{q}_{\text{CONST}}]$, in reply to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{3,(\omega-1)'}$.
- For $\theta \in [\hat{q}_{\text{DEL}}]$, in response to the θ^{th} delegated key query of $\mathcal{A}$ corresponding to TM pair $(M^{(\theta)}, \widetilde{M}^{(\theta)}) \in \mathbb{M}_\lambda^2$ with $[M^{(\theta)}(x^*) = 1] \wedge [\widetilde{M}^{(\theta)}(x^*) = 0]$, if $\theta \neq \omega$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{3,(\omega-1)'}$, while if $\theta = \omega$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects the PPRF keys $K_1^{(\omega)}, \ldots, K_\lambda^{(\omega)}, K_{\text{SPS},A}^{(\omega)}, K_{\text{SPS},E}^{(\omega)}, \widetilde{K}'^{(\omega)}, \widetilde{K}_1^{(\omega)}, \ldots, \widetilde{K}_\lambda^{(\omega)}, \widetilde{K}_{\text{SPS},A}^{(\omega)}, \widetilde{K}_{\text{SPS},E}^{(\omega)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. After that, it generates $(\text{PP}_{\text{ACC}}^{(\omega)}, w_0^{(\omega)}, \text{STORE}_0^{(\omega)}), (\widetilde{\text{PP}}_{\text{ACC}}^{(\omega)}, \widetilde{w}_0^{(\omega)}, \widetilde{\text{STORE}}_0^{(\omega)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}_{\text{ITR}}^{(\omega)}, v_0^{(\omega)}), (\widetilde{\text{PP}}_{\text{ITR}}^{(\omega)}, \widetilde{v}_0^{(\omega)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. Next, $\mathcal{B}$ sends (h^*, ℓ^*) as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back a punctured PPRF key $K^*\{(h^*, \ell^*)\}$ and a value $r^* = r_1^* \| r_2^* \in \mathcal{Y}_{\text{PPRF}}$, where either $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$ or $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$. $\mathcal{B}$ *implicitly* views the key K^* as the key $K'^{(\omega)}$.
  4. Then, $\mathcal{B}$ forms $(\text{PK}_{\text{PKE}}^{*(\omega)}, \text{SK}_{\text{PKE}}^{*(\omega)}) = \mathsf{PKE.Setup}(1^\lambda;\ r_1^*)$ and forms $\text{CT}_{\text{PKE}}^{*(\omega)} = \mathsf{PKE.Encrypt}(\text{PK}_{\text{PKE}}^{*(\omega)}, \mathcal{F}(K, (h^*, \ell^*));\ r_2^*)$.
  5. $\mathcal{B}$ gives $\mathcal{A}$ the delegated key $\text{SK}_{\text{DCPRF}}\{M^{(\omega)} \wedge \widetilde{M}^{(\omega)}\} = (\widetilde{K}'^{(\omega)}, \text{HK}, \text{PP}_{\text{ACC}}^{(\omega)}, \widetilde{\text{PP}}_{\text{ACC}}^{(\omega)}, w_0^{(\omega)}, \widetilde{w}_0^{(\omega)}, \text{STORE}_0^{(\omega)}, \widetilde{\text{STORE}}_0^{(\omega)}, \text{PP}_{\text{ITR}}^{(\omega)}, \widetilde{\text{PP}}_{\text{ITR}}^{(\omega)}, v_0^{(\omega)}, \widetilde{v}_0^{(\omega)}, \mathcal{P}_1^{(\omega)}, \widetilde{\mathcal{P}}_1^{(\omega)}, \mathcal{P}_2^{(\omega)}, \widetilde{\mathcal{P}}_2^{(\omega)}, \mathcal{P}_3^{(\omega)}, \widetilde{\mathcal{P}}_3^{(\omega)}, \mathcal{P}_{\text{DCPRF}}^{(\omega)}, \widetilde{\mathcal{P}}_{\text{DCPRF}}^{(\omega)})$, where
     - $\mathcal{P}_1^{(\omega)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\omega)}, w_0^{(\omega)}, v_0^{(\omega)}, K_{\text{SPS},E}^{(\omega)}])$,
     - $\widetilde{\mathcal{P}}_1^{(\omega)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[\widetilde{q}_0^{(\omega)}, \widetilde{w}_0^{(\omega)}, \widetilde{v}_0^{(\omega)}, \widetilde{K}_{\text{SPS},E}^{(\omega)}])$,
     - $\mathcal{P}_2^{(\omega)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\omega)}, \text{PP}_{\text{ITR}}^{(\omega)}, K_{\text{SPS},E}^{(\omega)}])$,
     - $\widetilde{\mathcal{P}}_2^{(\omega)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \widetilde{\text{PP}}_{\text{ACC}}^{(\omega)}, \widetilde{\text{PP}}_{\text{ITR}}^{(\omega)}, \widetilde{K}_{\text{SPS},E}^{(\omega)}])$,
     - $\mathcal{P}_3^{(\omega)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\omega)} K_{\text{SPS},E}^{(\omega)}])$,
     - $\widetilde{\mathcal{P}}_3^{(\omega)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[\widetilde{K}_{\text{SPS},A}^{(\omega)}, \widetilde{K}_{\text{SPS},E}^{(\omega)}])$,
     - $\mathcal{P}_{\text{DCPRF}}^{(\omega)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}''_{\text{DCPRF}}[M^{(\omega)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\omega)}, \text{PP}_{\text{ITR}}^{(\omega)}, K, K^*\{(h^*, \ell^*)\}, K_1^{(\omega)}, \ldots, K_\lambda^{(\omega)}, K_{\text{SPS},A}^{(\omega)}, \text{CT}_{\text{PKE}}^{*(\omega)}, h^*, \ell^*])$,
     - $\widetilde{\mathcal{P}}_{\text{DCPRF}}^{(\omega)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\text{DCPRF}}[\widetilde{M}^{(\omega)}, T = 2^\lambda, \widetilde{\text{PP}}_{\text{ACC}}^{(\omega)}, \widetilde{\text{PP}}_{\text{ITR}}^{(\omega)}, K^*\{(h^*, \ell^*)\}, \widetilde{K}'^{(\omega)}, \widetilde{K}_1^{(\omega)}, \ldots, \widetilde{K}_\lambda^{(\omega)}, \widetilde{K}_{\text{SPS},A}^{(\omega)}, h^*, \ell^*])$.
- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its PPRF selective pseudorandomness experiment.

Note that if $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{3,(\omega-1)'}$. On the other hand, if $r^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{3,\omega}$. This completes the proof of Lemma C.4. □

Lemma C.5. *Assuming* PKE *is* CPA *secure, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ, $|\mathsf{Adv}_{\mathcal{A}}^{(3,\omega)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(3,\omega')}(\lambda)| \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl.

Proof. We will focus on the case where the parent TM $M^{(\omega)} \in \mathbb{M}_\lambda$ of the ω^{th} delegated key query $(M^{(\omega)}, \widetilde{M}^{(\omega)}) \in \mathbb{M}_\lambda^2$ made by $\mathcal{A}$ is not one that has appeared in some previous delegated key query of $\mathcal{A}$. This is because in the other case by the description of the hybrid experiments it readily follows that $|\mathsf{Adv}_{\mathcal{A}}^{(3,\omega)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(3,\omega')}(\lambda)| = 0$.

Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}_{\mathcal{A}}^{(3,\omega)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(3,\omega')}(\lambda)|$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the CPA security of PKE using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ receives a PKE public key PK^*_{PKE} from its PKE CPA security challenger $\mathcal{C}$. Then, $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x^*_0 \dots x^*_{\ell^*-1} \in \mathcal{X}_{\text{DCPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$.
  2. Next, $\mathcal{B}$ selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  3. Then $\mathcal{B}$ chooses a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, then $\mathcal{B}$ computes $y^* = \mathcal{F}(K, (h^*, \ell^*))$. Otherwise, it picks $y^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$.
  4. $\mathcal{B}$ returns the challenge DCPRF value y^* to $\mathcal{A}$.
- For $\eta \in [\hat{q}_{\text{CONST}}]$, in reply to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{3,\omega}$.
- For $\theta \in [\hat{q}_{\text{DEL}}]$, in response to the θ^{th} delegated key query of $\mathcal{A}$ corresponding to TM pair $(M^{(\theta)}, \widetilde{M}^{(\theta)}) \in \mathbb{M}^2_\lambda$ with $[M^{(\theta)}(x^*) = 1] \wedge [\widetilde{M}^{(\theta)}(x^*) = 0]$, if $\theta \neq \omega$, then $\mathcal{B}$ proceeds exactly as in $\mathsf{Hyb}_{3,\omega}$, while if $\theta = \omega$, then $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K'^{(\omega)}, K^{(\omega)}_1, \dots, K^{(\omega)}_\lambda, K^{(\omega)}_{\text{SPS},A}, K^{(\omega)}_{\text{SPS},E}, \widetilde{K}'^{(\omega)}, \widetilde{K}^{(\omega)}_1, \dots, \widetilde{K}^{(\omega)}_\lambda, \widetilde{K}^{(\omega)}_{\text{SPS},A}, \widetilde{K}^{(\omega)}_{\text{SPS},E} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. After that, it generates $(\text{PP}^{(\omega)}_{\text{ACC}}, w^{(\omega)}_0, \text{STORE}^{(\omega)}_0), (\widetilde{\text{PP}}^{(\omega)}_{\text{ACC}}, \widetilde{w}^{(\omega)}_0, \widetilde{\text{STORE}}^{(\omega)}_0) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\text{PP}^{(\omega)}_{\text{ITR}}, v^{(\omega)}_0), (\widetilde{\text{PP}}^{(\omega)}_{\text{ITR}}, \widetilde{v}^{(\omega)}_0) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\text{ITR}} = 2^\lambda)$.
  3. Then, $\mathcal{B}$ creates the punctured PPRF key $K'^{(\omega)}\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K'^{(\omega)}, (h^*, \ell^*))$.
  4. Next, $\mathcal{B}$ sends the two messages $\mathsf{MSG}_0 = \mathcal{F}(K, (h^*, \ell^*)) \in \mathcal{Y}_{\text{PPRF}}$ and $\mathsf{MSG}_1 = \hat{y}^{*(\omega)} \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}$ to $\mathcal{C}$ and receives back a PKE ciphertext CT^*_{PKE} from $\mathcal{C}$, where either $\text{CT}^*_{\text{PKE}} \xleftarrow{\$} \mathsf{PKE.Encrypt}(\text{PK}^*_{\text{PKE}}, \mathcal{F}(K, (h^*, \ell^*)))$ or $\text{CT}^*_{\text{PKE}} \xleftarrow{\$} \mathsf{PKE.Encrypt}(\text{PK}^*_{\text{PKE}}, \hat{y}^*\omega))$.
  5. $\mathcal{B}$ gives $\mathcal{A}$ the delegated key $\text{SK}_{\text{DCPRF}}\{M^{(\omega)} \wedge \widetilde{M}^{(\omega)}\} = (\widetilde{K}'^{(\omega)}, \text{HK}, \text{PP}^{(\omega)}_{\text{ACC}}, \widetilde{\text{PP}}^{(\omega)}_{\text{ACC}}, w^{(\omega)}_0, \widetilde{w}^{(\omega)}_0, \text{STORE}^{(\omega)}_0, \widetilde{\text{STORE}}^{(\omega)}_0, \text{PP}^{(\omega)}_{\text{ITR}}, \widetilde{\text{PP}}^{(\omega)}_{\text{ITR}}, v^{(\omega)}_0, \widetilde{v}^{(\omega)}_0, \mathcal{P}^{(\omega)}_1, \widetilde{\mathcal{P}}^{(\omega)}_1, \mathcal{P}^{(\omega)}_2, \widetilde{\mathcal{P}}^{(\omega)}_2, \mathcal{P}^{(\omega)}_3, \widetilde{\mathcal{P}}^{(\omega)}_3, \mathcal{P}^{(\omega)}_{\text{DCPRF}}, \widetilde{\mathcal{P}}^{(\omega)}_{\text{DCPRF}})$, where
     - $\mathcal{P}^{(\omega)}_1 = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q^{(\omega)}_0, w^{(\omega)}_0, v^{(\omega)}_0, K^{(\omega)}_{\text{SPS},E}])$,
     - $\widetilde{\mathcal{P}}^{(\omega)}_1 = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[\widetilde{q}^{(\omega)}_0, \widetilde{w}^{(\omega)}_0, \widetilde{v}^{(\omega)}_0, \widetilde{K}^{(\omega)}_{\text{SPS},E}])$,
     - $\mathcal{P}^{(\omega)}_2 = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}^{(\omega)}_{\text{ACC}}, \text{PP}^{(\omega)}_{\text{ITR}}, K^{(\omega)}_{\text{SPS},E}])$,
     - $\widetilde{\mathcal{P}}^{(\omega)}_2 = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \widetilde{\text{PP}}^{(\omega)}_{\text{ACC}}, \widetilde{\text{PP}}^{(\omega)}_{\text{ITR}}, \widetilde{K}^{(\omega)}_{\text{SPS},E}])$,
     - $\mathcal{P}^{(\omega)}_3 = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K^{(\omega)}_{\text{SPS},A}, K^{(\omega)}_{\text{SPS},E}])$,
     - $\widetilde{\mathcal{P}}^{(\omega)}_3 = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[\widetilde{K}^{(\omega)}_{\text{SPS},A}, \widetilde{K}^{(\omega)}_{\text{SPS},E}])$,
     - $\mathcal{P}^{(\omega)}_{\text{DCPRF}} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}''_{\text{DCPRF}}[M^{(\omega)}, T = 2^\lambda, \text{PP}^{(\omega)}_{\text{ACC}}, \text{PP}^{(\omega)}_{\text{ITR}}, K, K'^{(\omega)}\{(h^*, \ell^*)\}, K^{(\omega)}_1, \dots, K^{(\omega)}_\lambda, K^{(\omega)}_{\text{SPS},A}, \text{CT}^*_{\text{PKE}}, h^*, \ell^*])$,
     - $\widetilde{\mathcal{P}}^{(\omega)}_{\text{DCPRF}} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\text{DCPRF}}[\widetilde{M}^{(\omega)}, T = 2^\lambda, \widetilde{\text{PP}}^{(\omega)}_{\text{ACC}}, \widetilde{\text{PP}}^{(\omega)}_{\text{ITR}}, K'^{(\omega)}\{(h^*, \ell^*)\}, \widetilde{K}'^{(\omega)}, \widetilde{K}^{(\omega)}_1, \dots, \widetilde{K}^{(\omega)}_\lambda, \widetilde{K}^{(\omega)}_{\text{SPS},A}, h^*, \ell^*])$.
- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its PKE CPA security experiment.

Note that if $\text{CT}^*_{\text{PKE}} \xleftarrow{\$} \mathsf{PKE.Encrypt}(\text{PK}^*_{\text{PKE}}, \mathcal{F}(K, (h^*, \ell^*)))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{3,\omega}$. On the other hand, if $\text{CT}^*_{\text{PKE}} \xleftarrow{\$} \mathsf{PKE.Encrypt}(\text{PK}^*_{\text{PKE}}, \hat{y}^{*(\omega)})$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{3,\omega'}$. This completes the proof of Lemma C.5. □

Lemma C.6. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$ satisfies the correctness under puncturing property described in Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(3,\hat{q}'_{\text{DEL}})}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(4)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The difference between $\mathsf{Hyb}_{3,\hat{q}'_{\mathrm{DEL}}}$ and Hyb_4 are the following:

(I) For $\eta \in [\hat{q}_{\mathrm{CONST}}]$, in $\mathsf{Hyb}_{3,\hat{q}'_{\mathrm{DEL}}}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0^{(\eta)})$ within the η^{th} constrained key provided to $\mathcal{A}$, while in Hyb_4, $\mathcal{B}$ includes the program $\mathcal{IO}(P_1^{(\eta)})$ instead, where
- $P_0^{(\eta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{DCPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K'^{(\eta)}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]$,
- $P_1^{(\eta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{DCPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K\{(h^*, \ell^*)\}, K'^{(\eta)}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]$,

the program $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{DCPRF}}$ being shown in Fig. C.1.

(II) For $\theta \in [\hat{q}_{\mathrm{DEL}}]$, within the θ^{th} delegated key returned to $\mathcal{A}$, $\mathcal{B}$ includes the program $\mathcal{IO}(P_0^{\prime(\theta)})$ in $\mathsf{Hyb}_{3,\hat{q}'_{\mathrm{DEL}}}$, whereas, in Hyb_4, $\mathcal{B}$ includes the program $\mathcal{IO}(P_1^{\prime(\theta)})$ instead, where
- $P_0^{\prime(\theta)} = \mathsf{Constrained\text{-}Key.Prog}''_{\mathrm{DCPRF}}[M^{(\theta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\theta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\theta)}, K, K'^{(\theta)}\{(h^*, \ell^*)\}, K_1^{(\theta)}, \ldots, K_\lambda^{(\theta)}, K_{\mathrm{SPS},A}^{(\theta)}, \mathrm{CT}_{\mathrm{PKE}}^{*(\theta)}, h^*, \ell^*]$,
- $P_1^{\prime(\theta)} = \mathsf{Constrained\text{-}Key.Prog}''_{\mathrm{DCPRF}}[M^{(\theta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\theta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\theta)}, K\{(h^*, \ell^*)\}, K'^{(\theta)}\{(h^*, \ell^*)\}, K_1^{(\theta)}, \ldots, K_\lambda^{(\theta)}, K_{\mathrm{SPS},A}^{(\theta)}, \mathrm{CT}_{\mathrm{PKE}}^{*(\theta)}, h^*, \ell^*]$,

the program $\mathsf{Constrained\text{-}Key.Prog}''_{\mathrm{DCPRF}}$ being shown in Fig. C.2.

We will argue that the programs $P_0^{(\eta)}$ and $P_1^{(\eta)}$, as well as, the programs $P_0^{\prime(\theta)}$ and $P_1^{\prime(\theta)}$ are functionally equivalent, so that, by the security of $\mathcal{IO}$, Lemma C.6 follows. The programs $P_0^{(\eta)}$ and $P_1^{(\eta)}$ are functionally equivalent as both programs output $\perp$ for inputs corresponding to (h^*, ℓ^*) and by the correctness under puncturing property of the PPRF $\mathcal{F}$ the programs clearly have the same outputs for inputs corresponding to $(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)$. The programs $P_0^{\prime(\theta)}$ and $P_1^{\prime(\theta)}$ are also functionally identical as for inputs corresponding to (h^*, ℓ^*), both the programs output the same hardwired ciphertext and for inputs corresponding to $(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)$, their outputs are the same again by the correctness under puncturing property of the PPRF $\mathcal{F}$. Ofcourse, to arrive at the result we would have to consider a sequence of intermediate hybrid experiments where in each hybrid experiment we switch the programs one at a time. □

Lemma C.7. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $\mathsf{Adv}_{\mathcal{A}}^{(4)}(\lambda) \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $\mathsf{Adv}_{\mathcal{A}}^{(4)}(\lambda)$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine. The description of $\mathcal{B}$ follows:

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge input $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{X}_{\mathrm{DCPRF}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- Upon receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.
  2. After that, $\mathcal{B}$ sends (h^*, ℓ^*) as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back a punctured PPRF key $K^*\{(h^*, \ell^*)\}$ and a value $r^* \in \mathcal{Y}_{\mathrm{PPRF}}$, where either $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$ or $r^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$. $\mathcal{B}$ *implicitly* views the key K^* as the key K.
  3. $\mathcal{B}$ returns the challenge DCPRF value $y^* = r^*$ to $\mathcal{A}$.
- For $\eta = 1, \ldots, \hat{q}_{\mathrm{CONST}}$, in reply to the η^{th} constrained key query of $\mathcal{A}$ corresponding to TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ selects PPRF keys $K'^{(\eta)}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.

2. Next, it creates $(\mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$.
3. $\mathcal{B}$ gives $\mathcal{A}$ the constrained key $\mathrm{SK}_{\mathrm{DCPRF}}\{M^{(\eta)}\} = (K'^{(\eta)}, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \mathcal{P}_1^{(\eta)}, \mathcal{P}_2^{(\eta)}, \mathcal{P}_3^{(\eta)}, \mathcal{P}_{\mathrm{DCPRF}}^{(\eta)})$, were
   - $\mathcal{P}_1^{(\eta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}])$,
   - $\mathcal{P}_2^{(\eta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}])$,
   - $\mathcal{P}_3^{(\eta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}])$,
   - $\mathcal{P}_{\mathrm{DCPRF}}^{(\eta)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{DCPRF}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K^*\{(h^*, \ell^*)\}, K'^{(\eta)}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*])$.

- For $\theta = 1, \ldots, \hat{q}_{\mathrm{DEL}}$, in response to the θ^{th} delegated key query of $\mathcal{A}$ corresponding to TM pair $(M^{(\theta)}, \widetilde{M}^{(\theta)}) \in \mathbb{M}_\lambda^2$ with $[M^{(\theta)}(x^*) = 1] \wedge [\widetilde{M}^{(\theta)}(x^*) = 0]$, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first selects PPRF keys $K'^{(\theta)}, K_1^{(\theta)}, \ldots, K_\lambda^{(\theta)}, K_{\mathrm{SPS},A}^{(\theta)}, K_{\mathrm{SPS},E}^{(\theta)}, \widetilde{K}'^{(\theta)}, \widetilde{K}_1^{(\theta)}, \ldots, \widetilde{K}_\lambda^{(\theta)}, \widetilde{K}_{\mathrm{SPS},A}^{(\theta)}, \widetilde{K}_{\mathrm{SPS},E}^{(\theta)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. After that, it generates $(\mathrm{PP}_{\mathrm{ACC}}^{(\theta)}, w_0^{(\theta)}, \mathrm{STORE}_0^{(\theta)}), (\widetilde{\mathrm{PP}}_{\mathrm{ACC}}^{(\theta)}, \widetilde{w}_0^{(\theta)}, \widetilde{\mathrm{STORE}}_0^{(\theta)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\mathrm{PP}_{\mathrm{ITR}}^{(\theta)}, v_0^{(\theta)}), (\widetilde{\mathrm{PP}}_{\mathrm{ITR}}^{(\theta)}, \widetilde{v}_0^{(\theta)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$.
  3. Next, $\mathcal{B}$ forms the punctured PPRF key $K'^{(\theta)}\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K'^{(\theta)}, (h^*, \ell^*))$.
  4. After that, $\mathcal{B}$ creates $(\mathrm{PK}_{\mathrm{PKE}}^{*(\theta)}, \mathrm{SK}_{\mathrm{PKE}}^{*(\theta)}) \xleftarrow{\$} \mathsf{PKE.Setup}(1^\lambda)$, selects $\hat{y}^{*(\theta)} \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$, and forms $\mathrm{CT}_{\mathrm{PKE}}^{*(\theta)} \xleftarrow{\$} \mathsf{PKE.Encrypt}(\mathrm{PK}_{\mathrm{PKE}}^{*(\theta)}, \hat{y}^{*(\theta)})$.
  5. $\mathcal{B}$ gives $\mathcal{A}$ the delegated key $\mathrm{SK}_{\mathrm{DCPRF}}\{M^{(\theta)} \wedge \widetilde{M}^{(\theta)}\} = (\widetilde{K}'^{(\theta)}, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\theta)}, \widetilde{\mathrm{PP}}_{\mathrm{ACC}}^{(\theta)}, w_0^{(\theta)}, \widetilde{w}_0^{(\theta)}, \mathrm{STORE}_0^{(\theta)}, \widetilde{\mathrm{STORE}}_0^{(\theta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\theta)}, \widetilde{\mathrm{PP}}_{\mathrm{ITR}}^{(\theta)}, v_0^{(\theta)}, \widetilde{v}_0^{(\theta)}, \mathcal{P}_1^{(\theta)}, \widetilde{\mathcal{P}}_1^{(\theta)}, \mathcal{P}_2^{(\theta)}, \widetilde{\mathcal{P}}_2^{(\theta)}, \mathcal{P}_3^{(\theta)}, \widetilde{\mathcal{P}}_3^{(\theta)}, \mathcal{P}_{\mathrm{DCPRF}}^{(\theta)}, \widetilde{\mathcal{P}}_{\mathrm{DCPRF}}^{(\theta)})$, where
     - $\mathcal{P}_1^{(\theta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\theta)}, w_0^{(\theta)}, v_0^{(\theta)}, K_{\mathrm{SPS},E}^{(\theta)}])$,
     - $\widetilde{\mathcal{P}}_1^{(\theta)} = \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[\widetilde{q}_0^{(\theta)}, \widetilde{w}_0^{(\theta)}, \widetilde{v}_0^{(\theta)}, \widetilde{K}_{\mathrm{SPS},E}^{(\theta)}])$,
     - $\mathcal{P}_2^{(\theta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\theta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\theta)}, K_{\mathrm{SPS},E}^{(\theta)}])$,
     - $\widetilde{\mathcal{P}}_2^{(\theta)} = \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \widetilde{\mathrm{PP}}_{\mathrm{ACC}}^{(\theta)}, \widetilde{\mathrm{PP}}_{\mathrm{ITR}}^{(\theta)}, \widetilde{K}_{\mathrm{SPS},E}^{(\theta)}])$,
     - $\mathcal{P}_3^{(\theta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\theta)}, K_{\mathrm{SPS},E}^{(\theta)}])$,
     - $\widetilde{\mathcal{P}}_3^{(\theta)} = \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[\widetilde{K}_{\mathrm{SPS},A}^{(\theta)}, \widetilde{K}_{\mathrm{SPS},E}^{(\theta)}])$,
     - $\mathcal{P}_{\mathrm{DCPRF}}^{(\theta)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}''_{\mathrm{DCPRF}}[M^{(\theta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\theta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\theta)}, K^*\{(h^*, \ell^*)\}, K'^{(\theta)}\{(h^*, \ell^*)\}, K_1^{(\theta)}, \ldots, K_\lambda^{(\theta)}, K_{\mathrm{SPS},A}^{(\theta)}, \mathrm{CT}_{\mathrm{PKE}}^{*(\theta)}, h^*, \ell^*])$,
     - $\widetilde{\mathcal{P}}_{\mathrm{DCPRF}}^{(\theta)} = \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{DCPRF}}[\widetilde{M}^{(\theta)}, T = 2^\lambda, \widetilde{\mathrm{PP}}_{\mathrm{ACC}}^{(\theta)}, \widetilde{\mathrm{PP}}_{\mathrm{ITR}}^{(\theta)}, K'^{(\theta)}\{(h^*, \ell^*)\}, \widetilde{K}'^{(\theta)}, \widetilde{K}_1^{(\theta)}, \ldots, \widetilde{K}_\lambda^{(\theta)}, \widetilde{K}_{\mathrm{SPS},A}^{(\theta)}, h^*, \ell^*])$.

     Ofcourse, once $\mathcal{B}$ generates the components pertaining to some parent TM $M^{(\theta)} \in \mathbb{M}_\lambda$ while answering to the θ^{th} delegated key query of $\mathcal{A}$, it reuses those components in all subsequent delegated key queries of $\mathcal{A}$ with the same parent TM $M^{(\theta)}$.
- At the end of interaction, $\mathcal{A}$ outputs a guess bit $b' \in \{0, 1\}$. $\mathcal{B}$ outputs $\hat{b}' = b'$ as its guess bit in its PPRF selective pseudorandomness experiment.

Observe that the simulation of Hyb_4 by $\mathcal{B}$ is perfect. Furter, if $\mathcal{A}$ wins in this simulated Hyb_4, then $\mathcal{B}$ wins in the selective pseudorandom experiment against the PPRF $\mathcal{F}$. This completes the proof of Lemma C.7. □

Appendix D: Proof of Theorem 6.1

Theorem 6.1 (Security of the ABS Scheme of Section 6.2). *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* $\mathsf{P/poly}$*, $\mathcal{F}$ is a secure puncturable pseudorandom function as*

per Definition 2.2, SSB *is a somewhere statistically binding hash function according to Definition 2.3,* ACC *is a secure positional accumulator as defined in Definition 2.4,* ITR *is a secure cryptographic iterator as per Definition 2.5,* SPS *is a secure splittable signature scheme according to Definition 2.6,* PRG *is a secure injective pseudorandom generator, and* SIG *is existentially unforgeable against chosen message attack, the* ABS *scheme of Section 6.2 satisfies all the criteria of a secure* ABS *defined in Definition 6.1.*

Proof.

► **Signer Privacy**: Note that for any message $\mathsf{msg} \in \mathcal{M}_{\text{ABS}}$, $(\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K])), \text{MSK}_{\text{ABS}} = (K, \text{HK})) \xleftarrow{\$} \mathsf{ABS.Setup}(1^\lambda)$, and $x \in \mathcal{U}_{\text{ABS}}$ with $|x| = \ell_x$, a signature on msg under x is of the form $\sigma_{\text{ABS}} = (\text{VK}_{\text{SIG}}, \sigma_{\text{SIG}})$, where $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda;\ \mathcal{F}(K, (\mathcal{H}_{\text{HK}}(x), \ell_x)))$, $\sigma_{\text{SIG}} = \mathsf{SIG.Sign}(\text{SK}_{\text{SIG}}, \mathsf{msg})$. Here, $\text{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$. Thus, the distribution of the signature σ_{ABS} is clearly the same regardless of the signing key $\text{SK}_{\text{ABS}}(M)$ that is used to compute it.

► **Existential Unforgeability**: We will prove the existential unforgeability of the ABS construction of Section 6.2 against selective attribute adaptive chosen message attack by means of a sequence of hybrid experiments. We will demonstrate based on the security of various primitives that the advantage of any PPT adversary $\mathcal{A}$ in consecutive hybrid experiments differs only negligibly as well as that in the final hybrid experiment is negligible. We note that due to the selective attribute setting, the challenger $\mathcal{B}$ knows the challenge attribute string $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{U}_{\text{ABS}}$ and the SSB hash value $h^* = \mathcal{H}_{\text{HK}}(x^*)$ before receiving any signing key or signature query from the adversary $\mathcal{A}$. Suppose, the total number of signing key query and signature query made by the adversary $\mathcal{A}$ be $\hat{q}_{\text{KEY}}$ and $\hat{q}_{\text{SIGN}}$ respectively. As noted in Remark 6.1, without loss of generality we will assume that $\mathcal{A}$ only queries signatures on messages under the challenge attribute string x^*. The description of the hybrid experiments follows:

Sequence of Hybrid Experiments

Hyb_0: This experiment corresponds to the real selective attribute adaptive chosen message unforgeability experiment described in Definition 6.1 of Section 6.1. More precisely, this experiment proceeds as follows:

- $\mathcal{A}$ submits a challenge attribute string $x^* = x_0^* \ldots x_{\ell^*-1}^* \in \mathcal{U}_{\text{ABS}}$ with $|x^*| = \ell^*$ to $\mathcal{B}$.
- $\mathcal{B}$ generates $(\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}_{\text{ABS}}[K])), \text{MSK}_{\text{ABS}} = (\text{HK}, K)) \xleftarrow{\$} \mathsf{ABS.Setup}(1^\lambda)$, as described in Section 6.2, and provides PP_{ABS} to $\mathcal{A}$.
- For $\eta = 1, \ldots, \hat{q}_{\text{KEY}}$, in response to the η^{th} signing key query corresponding to signing policy TM $M^{(\eta)} = \langle Q^{(\eta)}, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta^{(\eta)}, q_0^{(\eta)}, q_{\text{AC}}^{(\eta)}, q_{\text{REJ}}^{(\eta)} \rangle \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ creates

$$\text{SK}_{\text{ABS}}(M^{(\eta)}) = \left(\begin{array}{l} \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\text{ABS}}[M^{(\eta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\text{SPS},A}^{(\eta)}]) \end{array} \right)$$
$$\xleftarrow{\$} \mathsf{ABS.KeyGen}(\text{MSK}_{\text{ABS}}, M^{(\eta)}),$$

  as described in Section 6.2 and returns $\text{SK}_{\text{ABS}}(M^{(\eta)})$ to $\mathcal{A}$.
- For $\theta = 1, \ldots, \hat{q}_{\text{SIGN}}$, in reply to the θ^{th} signature query on message $\mathsf{msg}^{(\theta)}$ under attribute string x^*, $\mathcal{B}$ identifies some TM $M^* \in \mathbb{M}_\lambda$ such that $M^*(x^*) = 1$, generates $\text{SK}_{\text{ABS}}(M^*) \xleftarrow{\$}$

$\mathsf{ABS.KeyGen}(\mathrm{MSK}_{\mathrm{ABS}}, M^*)$, and computes $\sigma_{\mathrm{ABS}}^{(\theta)} = (\mathrm{VK}_{\mathrm{SIG}}^*, \sigma_{\mathrm{SIG}}^{(\theta)}) \xleftarrow{\$} \mathsf{ABS.Sign}(\mathrm{SK}_{\mathrm{ABS}}(M^*), x^*, \mathsf{msg}^{(\theta)})$ as described in Section 6.2. $\mathcal{B}$ gives back $\sigma_{\mathrm{ABS}}^{(\theta)}$ to $\mathcal{A}$.

- Finally, $\mathcal{A}$ outputs a forged signature σ_{ABS}^* on some message msg^* under attribute string x^*.

$\mathsf{Hyb}_{0,\nu}$ $(\nu = 1, \ldots, \hat{q}_{\mathrm{KEY}})$: This experiment is similar to Hyb_0 except that for $\eta \in [\hat{q}_{\mathrm{KEY}}]$, in reply to the η^{th} signing key query of $\mathcal{A}$ corresponding to signing policy TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ returns the signing key

$$\mathrm{SK}_{\mathrm{ABS}}(M^{(\eta)}) = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\underline{\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]}) \end{pmatrix},$$

if $\underline{\eta \leq \nu}$, where the program $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}$ is an alteration of the program $\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}$ (Fig. 6.2) and is described in Fig. D.1, while it returns the signing key

$$\mathrm{SK}_{\mathrm{ABS}}(M^{(\eta)}) = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}]) \end{pmatrix},$$

if $\underline{\eta > \nu}$. Observe that $\mathsf{Hyb}_{0,0}$ coincides with Hyb_0.

Hyb_1: This experiment coincides with $\mathsf{Hyb}_{0,\hat{q}_{\mathrm{KEY}}}$. More formally, in this experiment for $\eta = 1, \ldots, \hat{q}_{\mathrm{KEY}}$, in reply to the η^{th} signing key query of $\mathcal{A}$ corresponding to signing policy TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ generates all the components of the signing key as in Hyb_0, however, it returns the signing key

$$\mathrm{SK}_{\mathrm{ABS}}(M^{(\eta)}) = \begin{pmatrix} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]) \end{pmatrix}.$$

The rest of the experiment is analogous to Hyb_0.

Hyb_2: This experiment is identical to Hyb_1 other than the following exceptions:

(I) Upon receiving the challenge attribute string x^*, $\mathcal{B}$ proceeds as follows:

1. It selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$ and generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ just as in Hyb_1,

Constants: TM $M = \langle Q, \Sigma_{\text{INP}}, \Sigma_{\text{TAPE}}, \delta, q_0, q_{\text{AC}}, q_{\text{REJ}} \rangle$, Time bound $T = 2^\lambda$, Public parameters for positional accumulator PP_{ACC}, Public parameters for iterator PP_{ITR}, PPRF keys $K, K_1, \ldots, K_\lambda, K_{\text{SPS},A}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*

Inputs: Time t, String SEED_{IN}, Header position POS_{IN}, Symbol SYM_{IN}, TM state ST_{IN}, Accumulator value w_{IN}, Accumulator proof π_{ACC}, Auxiliary value AUX, Iterator value v_{IN}, SSB hash value h, Length ℓ_{INP}, Signature $\sigma_{\text{SPS,IN}}$

Output: (SIG signing key SK_{SIG}, SIG verification key VK_{SIG}), or Header Position (POS_{OUT}, Symbol SYM_{OUT}, TM state ST_{OUT}, Accumulator value w_{OUT}, Iterator value v_{OUT}, Signature $\sigma_{\text{SPS,OUT}}$, String SEED_{OUT}), or $\perp$

1. Identify an integer τ such that $2^\tau \leq t < 2^{\tau+1}$.
   If $[\mathsf{PRG}(\text{SEED}_{\text{IN}}) \neq \mathsf{PRG}(\mathcal{F}(K_\tau, (h, \ell_{\text{INP}})))] \ \wedge \ [t > 1]$, output $\perp$.
2. If $\mathsf{ACC.Verify\text{-}Read}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{IN}}, \text{POS}_{\text{IN}}, \pi_{\text{ACC}}) = 0$, output $\perp$.
3.(a) Compute $r_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t-1))$, $(\text{SK}_{\text{SPS},A}, \text{VK}_{\text{SPS},A}, \text{VK}_{\text{SPS-REJ},A}) = \mathsf{SSB.Setup}(1^\lambda;\ r_{\text{SPS},A})$.
   (b) Set $m_{\text{IN}} = (v_{\text{IN}}, \text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}})$. If $\mathsf{SPS.Verify}(\text{VK}_{\text{SPS},A}, m_{\text{IN}}, \sigma_{\text{SPS,IN}}) = 0$, output $\perp$.
4.(a) Compute $(\text{ST}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \beta) = \delta(\text{ST}_{\text{IN}}, \text{SYM}_{\text{IN}})$ and $\text{POS}_{\text{OUT}} = \text{POS}_{\text{IN}} + \beta$.
   (b) If $\text{ST}_{\text{OUT}} = q_{\text{REJ}}$, output $\perp$.
   Else if $[\text{ST}_{\text{OUT}} = q_{\text{AC}}] \ \wedge \ [(h, \ell_{\text{INP}}) \neq (h^*, \ell^*)]$, perform the following:
   (I) Compute $r_{\text{SIG}} = \mathcal{F}(K, (h, \ell_{\text{INP}}))$, $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda;\ r_{\text{SIG}})$.
   (II) Output $(\text{SK}_{\text{SIG}}, \text{VK}_{\text{SIG}})$.
   Else if $\text{ST}_{\text{OUT}} = q_{\text{AC}}$, output $\perp$.
5.(a) Compute $w_{\text{OUT}} = \mathsf{ACC.Update}(\text{PP}_{\text{ACC}}, w_{\text{IN}}, \text{SYM}_{\text{OUT}}, \text{POS}_{\text{IN}}, \text{AUX})$. If $w_{\text{OUT}} = \perp$, output $\perp$.
   (b) Compute $v_{\text{OUT}} = \mathsf{ITR.Iterate}(\text{PP}_{\text{ITR}}, v_{\text{IN}}, (\text{ST}_{\text{IN}}, w_{\text{IN}}, \text{POS}_{\text{IN}}))$.
6.(a) Compute $r'_{\text{SPS},A} = \mathcal{F}(K_{\text{SPS},A}, (h, \ell_{\text{INP}}, t))$, $(\text{SK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS},A}, \text{VK}'_{\text{SPS-REJ},A}) = \mathsf{SPS.Setup}(1^\lambda;\ r'_{\text{SPS},A})$.
   (b) Set $m_{\text{OUT}} = (v_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, \text{POS}_{\text{OUT}})$. Compute $\sigma_{\text{SPS,OUT}} = \mathsf{SPS.Sign}(\text{SK}'_{\text{SPS},A}, m_{\text{OUT}})$.
7. If $t + 1 = 2^{\tau'}$, set $\text{SEED}_{\text{OUT}} = \mathcal{F}(K_{\tau'}, (h, \ell_{\text{INP}}))$.
   Else, set $\text{SEED}_{\text{OUT}} = \epsilon$
8. Output $(\text{POS}_{\text{OUT}}, \text{SYM}_{\text{OUT}}, \text{ST}_{\text{OUT}}, w_{\text{OUT}}, v_{\text{OUT}}, \sigma_{\text{SPS,OUT}}, \text{SEED}_{\text{OUT}})$.

Fig. D.1. $\mathsf{Constrained\text{-}Key.Prog}'_{\text{ABS}}$

2. It then computes $h^* = \mathcal{H}_{\text{HK}}(x^*)$ and creates the punctured PPRF key $K\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K, (h^*, \ell^*))$,
3. It computes $\widehat{r}^*_{\text{SIG}} = \mathcal{F}(K, (h^*, \ell^*))$, forms $(\widehat{\text{SK}}^*_{\text{SIG}}, \widehat{\text{VK}}^*_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda; \widehat{r}^*_{\text{SIG}})$,
4. It sets the public parameters PP_{ABS} to be given to $\mathcal{A}$ as $\text{PP}_{\text{ABS}} = (\text{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\text{ABS}}[K\{(h^*, \ell^*)\}, \widehat{\text{VK}}^*_{\text{SIG}}, h^*, \ell^*]))$, where the program $\mathsf{Verify.Prog}'_{\text{ABS}}$ is an alteration of the program $\mathsf{Verify.Prog}_{\text{ABS}}$ (Fig. 6.1) and is depicted in Fig. D.2.

Constants: Punctured PPRF key $K\{(h^*, \ell^*)\}$, SIG verification key $\widehat{\text{VK}}^*_{\text{SIG}}$, SSB hash value of challenge input h^*, Length of challenge input ℓ^*
Inputs: SSB hash value h, Length ℓ_{INP}
Output: SIG verification key $\widehat{\text{VK}}_{\text{SIG}}$

(a) If $(h, \ell_{\text{INP}}) = (h^*, \ell^*)$, output $\widehat{\text{VK}}^*_{\text{SIG}}$.
Else compute $\hat{r}_{\text{SIG}} = \mathcal{F}(K\{(h^*, \ell^*)\}, (h, \ell_{\text{INP}}))$, $(\widehat{\text{SK}}_{\text{SIG}}, \widehat{\text{VK}}_{\text{SIG}}) = \mathsf{SIG.Setup}(1^\lambda;\ \hat{r}_{\text{SIG}})$.
(b) Output $\widehat{\text{VK}}_{\text{SIG}}$.

Fig. D.2. $\mathsf{Verify.Prog}'_{\text{ABS}}$

(II) For $\eta = 1, \ldots, \hat{q}_{\text{KEY}}$, in response to the η^{th} signing key query of $\mathcal{A}$ corresponding to signing policy TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ provides $\mathcal{A}$ with the signing key

$$\mathrm{SK}_{\text{ABS}}(M^{(\eta)}) = \begin{pmatrix} \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, w_0^{(\eta)}, \text{STORE}_0^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \text{HK}, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\text{SPS},A}^{(\eta)}, K_{\text{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\text{ABS}}[M^{(\eta)}, T = 2^\lambda, \text{PP}_{\text{ACC}}^{(\eta)}, \text{PP}_{\text{ITR}}^{(\eta)}, \underline{K\{(h^*, \ell^*)\}}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, \\ \qquad K_{\text{SPS},A}^{(\eta)}, h^*, \ell^*]) \end{pmatrix}.$$

Hyb_3: This experiment is similar to Hyb_2 with the only exception that $\mathcal{B}$ selects $\underline{\hat{r}_{\text{SIG}}^* \xleftarrow{\$} \mathcal{Y}_{\text{PPRF}}}$. More formally, this experiment has the following deviations from hyb_2:

(I) In this experiment $\mathcal{B}$ creates the punctured PPRF key $K\{(h^*, \ell^*)\}$ as in Hyb_2, however, it generates $\underline{(\widehat{\text{SK}}_{\text{SIG}}^*, \widehat{\text{VK}}_{\text{SIG}}^*) \xleftarrow{\$} \mathsf{SIG.Setup}(1^\lambda)}$. It includes the obfuscated program $\mathcal{IO}(\mathsf{Verify.Prog}'_{\text{ABS}}[K\{(h^*, \ell^*)\}, \widehat{\text{VK}}_{\text{SIG}}^*, h^*, \ell^*])$ within the public parameters PP_{ABS} to be provided to $\mathcal{A}$ as earlier.

(II) Also, for $\theta = 1, \ldots, \hat{q}_{\text{SIGN}}$, to answer the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\text{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\underline{\sigma_{\text{SIG}}^{(\theta)} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\text{SK}}_{\text{SIG}}^*, \mathsf{msg}^{(\theta)})}$ and returns $\underline{\sigma_{\text{ABS}}^{(\theta)} = (\widehat{\text{VK}}_{\text{SIG}}^*, \sigma_{\text{SIG}}^{(\theta)})}$ to $\mathcal{A}$.

Analysis

Let $\mathsf{Adv}_{\mathcal{A}}^{(0)}(\lambda), \mathsf{Adv}_{\mathcal{A}}^{(0,\nu)}(\lambda)$ $(\nu = 1, \ldots, \hat{q}_{\text{KEY}}), \mathsf{Adv}_{\mathcal{A}}^{(1)}(\lambda), \mathsf{Adv}_{\mathcal{A}}^{(2)}(\lambda)$, and $\mathsf{Adv}_{\mathcal{A}}^{(3)}(\lambda)$ represent respectively the advantage of the adversary $\mathcal{A}$, i.e., $\mathcal{A}$'s probability of successfully outputting a valid forgery, in $\mathsf{Hyb}_0, \mathsf{Hyb}_{0,\nu}$ $(\nu = 1, \ldots, \hat{q}_{\text{KEY}}), \mathsf{Hyb}_1$, Hyb_2, and Hyb_3 respectively. Then, by the description of the hybrid experiments it follows that $\mathsf{Adv}_{\mathcal{A}}^{\text{ABS,UF-CMA}}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,0)}(\lambda)$ and $\mathsf{Adv}_{\mathcal{A}}^{(1)}(\lambda) \equiv \mathsf{Adv}_{\mathcal{A}}^{(0,\hat{q}_{\text{KEY}})}(\lambda)$. Hence, we have

$$\mathsf{Adv}_{\mathcal{A}}^{\text{ABS,UF-CMA}}(\lambda) \leq \sum_{\nu=1}^{\hat{q}_{\text{KEY}}} |\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(0,\nu)}(\lambda)| + \sum_{j=1}^{2} |\mathsf{Adv}_{\mathcal{A}}^{(j)}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{(j+1)}(\lambda)| + \mathsf{Adv}_{\mathcal{A}}^{(3)}(\lambda). \tag{D.1}$$

Lemmas D.1–D.4 will show that the RHS of Eq. (D.1) is negligible and thus the existential unforgeability of the ABS construction of Section 6.2 follows. □

D.1 Lemmas for the Proof of Theorem 6.1

Lemma D.1. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for $\mathsf{P/poly}$, $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, SSB is a somewhere statistically binding hash function according to Definition 2.3, ACC is a secure positional accumulator as defined in Definition 2.4, ITR is a secure cryptographic iterator as per Definition 2.5, SPS is a secure splittable signature scheme according to Definition 2.6, and PRG is a secure injective pseudorandom generator, for any PPT adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}_{\mathcal{A}}^{(0,\nu-1)}(\lambda)\mathsf{Adv}_{\mathcal{A}}^{(0,\nu)}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function negl.*

Proof. The proof of Lemma B.1 is similar to that of Lemma A.3 and, therefore, is omitted to avoid repetition. □

Lemma D.2. *Assuming $\mathcal{IO}$ is a secure indistinguishability obfuscator for* P/poly *and $\mathcal{F}$ satisfies the correctness under puncturing property defined in Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(1)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. The two differences between Hyb_1 and Hyb_2 are the following:

(I) In Hyb_1, $\mathcal{B}$ includes $\mathcal{IO}(V_0)$ within the public parameters $\mathrm{PP}_{\mathrm{ABS}}$ provided to $\mathcal{A}$, whereas, in Hyb_2, $\mathcal{B}$ includes the program $\mathcal{IO}(V_1)$ within $\mathrm{PP}_{\mathrm{ABS}}$, where
- $(V_0) = \mathsf{Verify.Prog}_{\mathrm{ABS}}[K]$ (Fig. 6.1),
- $(V_1) = \mathsf{Verify.Prog}'_{\mathrm{ABS}}[K\{(h^*, \ell^*)\}, \widehat{\mathrm{VK}}^*_{\mathrm{ABS}}, h^*, \ell^*]$ (Fig. D.2).

(II) For $\eta = 1, \ldots, \hat{q}_{\mathrm{KEY}}$, the signing key $\mathrm{SK}_{\mathrm{ABS}}(M^{(\eta)})$ returned by $\mathcal{B}$ to $\mathcal{A}$ corresponding to signing policy TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, includes the program $\mathcal{IO}(P_0^{(\eta)})$ in the experiment Hyb_1, while $\mathrm{SK}_{\mathrm{ABS}}(M^{(\eta)})$ includes the program $\mathcal{IO}(P_1^{(\eta)})$ in Hyb_2, where
- $P_0^{(\eta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}^{(\eta)}_{\mathrm{ACC}}, \mathrm{PP}^{(\eta)}_{\mathrm{ITR}}, K, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K^{(\eta)}_{\mathrm{SPS},A}, h^*, \ell^*]$,
- $P_1^{(\eta)} = \mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}^{(\eta)}_{\mathrm{ACC}}, \mathrm{PP}^{(\eta)}_{\mathrm{ITR}}, K\{(h^*, \ell^*)\}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K^{(\eta)}_{\mathrm{SPS},A}, h^*, \ell^*]$,

the program $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}$ being described in Fig. D.1.

Now, observe that on input $(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)$, both the programs V_0 and V_1 operates in the same manner only that the latter one uses the punctured PPRF key $K\{(h^*, \ell^*)\}$ for computing the string $\hat{r}_{\mathrm{SIG}}$ instead of the full PPRF key K used by the former program. Therefore, by the correctness under puncturing property of PPRF $\mathcal{F}$, it follows that for all inputs $(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)$, both the programs have identical output. Moreover, on input (h^*, ℓ^*), V_1 outputs the hardwired SIG verification key $\widehat{\mathrm{VK}}^*_{\mathrm{SIG}}$ which is computed as $(\widehat{\mathrm{SK}}^*_{\mathrm{SIG}}, \widehat{\mathrm{VK}}^*_{\mathrm{SIG}}) = \mathsf{SIG.Setup}(1^\lambda;\ \hat{r}^*_{\mathrm{SIG}})$, where $\hat{r}^*_{\mathrm{SIG}} = \mathcal{F}(K, (h^*, \ell^*))$. Notice that these values are exactly the same as those outputted V_0 on input (h^*, ℓ^*). Thus, the two programs are functionally equivalent.

Further, note that the program $\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}$ computes $\mathcal{F}(K, (h, \ell_{\mathrm{INP}}))$ if and only if $(h, \ell_{\mathrm{INP}}) \neq (h^*, \ell^*)$. Thus, again by the correctness under puncturing property of PPRF $\mathcal{F}$, the programs $P_0^{(\eta)}$ and $P_1^{(\eta)}$ are functionally equivalent as well for all $\eta \in [\hat{q}_{\mathrm{KEY}}]$.

Thus the security of $\mathcal{IO}$, Lemma D.2 follows. Observe that to prove this lemma we would actually have to proceed through a sequence of intermediate hybrid experiments where in each hybrid experiment we switch the programs one at a time. □

Lemma D.3. *Assuming $\mathcal{F}$ is a secure puncturable pseudorandom function as per Definition 2.2, for any* PPT *adversary $\mathcal{A}$, for any security parameter λ, $|\mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$ for some negligible function* negl.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ for which $|\mathsf{Adv}^{(2)}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda)|$ is non-negligible. Below we construct a PPT adversary $\mathcal{B}$ that breaks the selective pseudorandomness of the PPRF $\mathcal{F}$ using $\mathcal{A}$ as a sub-routine.

- $\mathcal{B}$ initializes $\mathcal{A}$ on input 1^λ and receives a challenge attribute string $x^* = x^*_0 \ldots x^*_{\ell^*-1} \in \mathcal{U}_{\mathrm{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- After receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.
  2. $\mathcal{B}$ sends (h^*, ℓ^*) as the challenge input to its PPRF selective pseudorandomness challenger $\mathcal{C}$ and receives back a punctured PPRF key $K^*\{(h^*, \ell^*)\}$ along with a challenge value $r^* \in \mathcal{Y}_{\mathrm{PPRF}}$, where either $r^* = \mathcal{F}(K^*, (h^*, \ell^*)\}$ or $r^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$. $\mathcal{B}$ *implicitly* views the key K^* as the key K.
  3. Then $\mathcal{B}$ creates $(\widehat{\mathrm{SK}}^*_{\mathrm{SIG}}, \widehat{\mathrm{VK}}^*_{\mathrm{SIG}}) = \mathsf{SIG.Setup}(1^\lambda;\ r^*)$.

4. Next, $\mathcal{B}$ sets public parameters $\mathrm{PP}_{\mathrm{ABS}} = (\mathrm{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\mathrm{ABS}}[K^*\{(h^*, \ell^*)\}, \widehat{\mathrm{VK}}^*_{\mathrm{SIG}}, h^*, \ell^*]))$ and gives it to $\mathcal{A}$.

- For $\eta = 1, \ldots, \hat{q}_{\mathrm{KEY}}$, to answer the η^{th} signing key query of $\mathcal{A}$ corresponding to signing policy TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:
  1. At first, $\mathcal{B}$ chooses PPRF keys $K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, $\mathcal{B}$ creates $(\mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$.
  3. $\mathcal{B}$ returns $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}(M^{(\eta)}) = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K^*\{(h^*, \ell^*)\}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, \\ \qquad K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]) \end{array} \right).$$

- For $\theta = 1, \ldots, \hat{q}_{\mathrm{SIGN}}$, in response to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\mathrm{ABS}}$ under attribute string x^*, $\mathcal{B}$ computes $\sigma_{\mathrm{SIG}}^{(\theta)} \xleftarrow{\$} \mathsf{SIG.Sign}(\widehat{\mathrm{SK}}^*_{\mathrm{SIG}}, \mathsf{msg}^{(\theta)})$ and provides $\mathcal{A}$ with $\sigma_{\mathrm{ABS}}^{(\theta)} = (\widehat{\mathrm{VK}}^*_{\mathrm{SIG}}, \sigma_{\mathrm{SIG}}^{(\theta)})$.
- Finally, $\mathcal{A}$ output a signature σ^*_{ABS} on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $\hat{b}' = 1$ as its guess bit in its PPRF selective pseudorandomness experiment if $\mathcal{A}$ wins, i.e., if $\mathsf{ABS.Verify}(\mathrm{PP}_{\mathrm{ABS}}, x^*, \mathsf{msg}^*, \sigma^*_{\mathrm{ABS}}) = 1$ and $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\mathrm{SIGN}}]$. Otherwise, $\mathcal{B}$ outputs $\hat{b}' = 0$ in its PPRF selective pseudorandomness experiment.

Notice that if $r^* = \mathcal{F}(K^*, (h^*, \ell^*))$, then $\mathcal{B}$ perfectly simulates hyb_2. On the other hand, if $r^* \xleftarrow{\$} \mathcal{Y}_{\mathrm{PPRF}}$, then $\mathcal{B}$ perfectly simulates Hyb_3. This completes the proof of Lemma D.3. □

Lemma D.4. *Assuming* SIG *is existentially unforgeable against* CMA*, for any* PPT *adversary* $\mathcal{A}$*, for any security parameter* λ*,* $\mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ *for some negligible function* negl*.*

Proof. Suppose that there exists a PPT adversary $\mathcal{A}$ for which $\mathsf{Adv}^{(3)}_{\mathcal{A}}(\lambda)$ is non-negligible. We construct a PPT adversary $\mathcal{B}$ that breaks the existential unforgeability of SIG using $\mathcal{A}$ as a sub-routin. The description $\mathcal{B}$ is as follows:

- $\mathcal{B}$ receives a SIG verification key $\mathrm{VK}^*_{\mathrm{SIG}}$ from its SIG existential unforgeability challenger $\mathcal{C}$. Then, $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ and receives a challenge attribute string $x^* = x^*_0 \ldots x^*_{\ell^*-1} \in \mathcal{U}_{\mathrm{ABS}}$ with $|x^*| = \ell^*$ from $\mathcal{A}$.
- After receiving x^*, $\mathcal{B}$ proceeds as follows:
  1. $\mathcal{B}$ first generates $\mathrm{HK} \xleftarrow{\$} \mathsf{SSB.Gen}(1^\lambda, n_{\text{SSB-BLK}} = 2^\lambda, i^* = 0)$ and computes $h^* = \mathcal{H}_{\mathrm{HK}}(x^*)$.
  2. Next, it selects a PPRF key $K \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$ and creates the punctured PPRF key $K\{(h^*, \ell^*)\} \xleftarrow{\$} \mathcal{F}.\mathsf{Puncture}(K, (h^*, \ell^*))$.
  3. Next, $\mathcal{B}$ sets public parameters $\mathrm{PP}_{\mathrm{ABS}} = (\mathrm{HK}, \mathcal{IO}(\mathsf{Verify.Prog}'_{\mathrm{ABS}}[K\{(h^*, \ell^*)\}, \mathrm{VK}^*_{\mathrm{SIG}}, h^*, \ell^*]))$ and gives it to $\mathcal{A}$.
- For $\eta = 1, \ldots, \hat{q}_{\mathrm{KEY}}$, to answer the η^{th} signing key query of $\mathcal{A}$ corresponding to signing policy TM $M^{(\eta)} \in \mathbb{M}_\lambda$ with $M^{(\eta)}(x^*) = 0$, $\mathcal{B}$ executes the following steps:
  1. At first, $\mathcal{B}$ chooses PPRF keys $K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)} \xleftarrow{\$} \mathcal{F}.\mathsf{Setup}(1^\lambda)$.
  2. Next, $\mathcal{B}$ creates $(\mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}) \xleftarrow{\$} \mathsf{ACC.Setup}(1^\lambda, n_{\text{ACC-BLK}} = 2^\lambda)$ and $(\mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}) \xleftarrow{\$} \mathsf{ITR.Setup}(1^\lambda, n_{\mathrm{ITR}} = 2^\lambda)$.

3. $\mathcal{B}$ returns $\mathcal{A}$ the signing key

$$\mathrm{SK}_{\mathrm{ABS}}(M^{(\eta)}) = \left(\begin{array}{l} \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, w_0^{(\eta)}, \mathrm{STORE}_0^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, v_0^{(\eta)}, \\ \mathcal{IO}(\mathsf{Init\text{-}SPS.Prog}[q_0^{(\eta)}, w_0^{(\eta)}, v_0^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Accumulate.Prog}[n_{\text{SSB-BLK}} = 2^\lambda, \mathrm{HK}, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Change\text{-}SPS.Prog}[K_{\mathrm{SPS},A}^{(\eta)}, K_{\mathrm{SPS},E}^{(\eta)}]) \\ \mathcal{IO}(\mathsf{Constrained\text{-}Key.Prog}'_{\mathrm{ABS}}[M^{(\eta)}, T = 2^\lambda, \mathrm{PP}_{\mathrm{ACC}}^{(\eta)}, \mathrm{PP}_{\mathrm{ITR}}^{(\eta)}, K\{(h^*, \ell^*)\}, K_1^{(\eta)}, \ldots, K_\lambda^{(\eta)}, \\ \qquad K_{\mathrm{SPS},A}^{(\eta)}, h^*, \ell^*]) \end{array} \right).$$

- For $\theta = 1, \ldots, \hat{q}_{\mathrm{SIGN}}$, in response to the θ^{th} signature query of $\mathcal{A}$ on message $\mathsf{msg}^{(\theta)} \in \mathcal{M}_{\mathrm{ABS}}$ under attribute string x^*, $\mathcal{B}$ forwards the message $\mathsf{msg}^{(\theta)}$ to $\mathcal{C}$ and receives back a signature $\sigma_{\mathrm{SIG}}^{(\theta)}$ on $\mathsf{msg}^{(\theta)}$ from $\mathcal{C}$. $\mathcal{B}$ provides, $\sigma_{\mathrm{ABS}}^{(\theta)} = (\mathrm{VK}_{\mathrm{SIG}}^*, \sigma_{\mathrm{SIG}}^{(\theta)})$ to $\mathcal{A}$.
- At the end of interaction, $\mathcal{A}$ outputs a signature $\sigma_{\mathrm{ABS}}^* = (\widehat{\mathrm{VK}}_{\mathrm{SIG}}^*, \sigma_{\mathrm{SIG}}^*)$ on some message msg^* under attribute string x^*. $\mathcal{B}$ outputs $(\mathsf{msg}^*, \sigma_{\mathrm{SIG}}^*)$ as a forgery in its existential unforgeability experiment against SIG.

Observe that the simulation of the experiment Hyb_3 by $\mathcal{B}$ is perfect. Now, if $\mathcal{A}$ wins in the above simulated experiment, then the following must hold simultaneously:

(I) $\mathsf{ABS.Verify}(\mathrm{PP}_{\mathrm{ABS}}, x^*, \mathsf{msg}^*, \sigma_{\mathrm{ABS}}^*) = 1$.
(II) $\mathsf{msg}^* \neq \mathsf{msg}^{(\theta)}$ for any $\theta \in [\hat{q}_{\mathrm{SIGN}}]$.

Note that $\mathsf{ABS.Verify}(\mathrm{PP}_{\mathrm{ABS}}, x^*, \mathsf{msg}^*, \sigma_{\mathrm{ABS}}^*) = 1$ implies $[\widehat{\mathrm{VK}}_{\mathrm{SIG}}^* = \mathrm{VK}_{\mathrm{SIG}}^*] \wedge [\mathsf{SIG.Verify}(\widehat{\mathrm{VK}}_{\mathrm{SIG}}^*, \mathsf{msg}^*, \sigma_{\mathrm{SIG}}^*) = 1]$, i.e., $\mathsf{SIG.Verify}(\mathrm{VK}_{\mathrm{SIG}}^*, \mathsf{msg}^*, \sigma_{\mathrm{SIG}}^*) = 1$. Further, notice that $\mathsf{msg}^{(\theta)}$, for $\theta \in [\hat{q}_{\mathrm{SIGN}}]$, are the only messages that $\mathcal{B}$ queried a signature on to $\mathcal{C}$. Thus, $(\mathsf{msg}^*, \sigma_{\mathrm{SIG}}^*)$ is indeed a valid forgery in the existential unforgeability experiment against SIG. □