# Resumable Zero-Knowledge for Circuits from Symmetric Key Primitives

Handong Zhang[1,2], Puwen Wei[1,2](✉), Haiyang Xue[3], Yi Deng[4,5], Jinsong Li[1,2], Wei Wang[1,2], and Guoxiao Liu[1,2]

[1] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China
[2] School of Cyber Science and Technology, Shandong University, Qingdao, China
`hdzhang@mail.sdu.edu.cn`, `pwei@sdu.edu.cn`, `jsli@mail.sdu.edu.cn`, `weiwangsdu@sdu.edu.cn`, `liuguoxiao@mail.sdu.edu.cn`
[3] The University of Hong Kong, Pokfulam, Hong Kong
`haiyangxc@gmail.com`
[4] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[5] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
`deng@iie.ac.cn`

**Abstract.** Consider the scenario that the prover and the verifier perform the zero-knowledge (ZK) proof protocol for the same statement multiple times sequentially, where each proof is modeled as a session. We focus on the problem of how to resume a ZK proof efficiently in such scenario. We introduce a new primitive called *resumable honest verifier zero-knowledge proof of knowledge* (resumable HVZKPoK) and propose a general construction of the resumable HVZKPoK for circuits based on the "MPC-in-the-head" paradigm, where the complexity of the resumed session is less than that of the original ZK proofs. To ensure the knowledge soundness for the resumed session, we identify a property called extractable decomposition. Interestingly, most block ciphers satisfy this property and the cost of resuming session can be reduced dramatically when the underlying circuits are implemented with block ciphers. As a direct application of our resumable HVZKPoK, we construct a post quantum secure stateful signature scheme, which makes Picnic3 suitable for blockchain protocol. Using the same parameter setting of Picnic3, the sign/verify time of our subsequent signatures can be reduced to 3.1%/3.3% of Picnic3 and the corresponding signature size can be reduced to 36%. Moreover, by applying a parallel version of our method to the well known Cramer, Damgård and Schoenmakers (CDS) transformation, we get a compressed one-out-of-$N$ proof for circuits, which can be further used to construct a ring signature from symmetric key primitives only. When the ring size is less than $2^4$, the size of our ring signature scheme is only about 1/3 of Katz et al.'s construction.

**Keywords:** Resumable · honest verifier zero-knowledge · MPC-in-the-head · stateful signature · ring signature · blockchain.

## 1 Introduction

Zero-knowledge (ZK) proofs [42,43] and their non-interactive form (NIZK) [14], which allow a prover to convince a verifier of a certain statement without revealing any additional information, are among the most fundamental and important cryptographic primitives. It is known that there exists ZK proof [42] for any NP language, while the resulting construction is rather inefficient. A lot of works have been done to propose efficient (NI)ZK proofs for arbitrary circuits or specific algebraic computation, e.g., zk-SNARKs [11,38], which have short proof for a statement. Some works focus on the efficient composition of ZK proofs for several statements [26,37,1]. Other works such as [54,47,17] investigate the batch ZK proofs, which enable many instances of the same relation to be proved and verified simultaneously. Amongst most of those constructions, the randomness and the related transcripts are "recycled" or compressed in *one* execution of the resulting ZK protocol in order to reduce the cost of computation or communication.

Notice that one common case of ZK proof, however, is proving the same statement repeatedly multiple times. For example, a user may be required to provide digital signatures on different messages periodically, where each signature can be thought of as one execution of the NIZK proof of knowledge of the signing key [9]. One typical application is validating the authenticity of firmware updates for IoT devices. The manufacturer periodically offers firmware updates and the corresponding signatures, and the IoT device needs to verify these signatures in order to ensure the authenticity of the updates. Another direct application of ZK is the identification protocol, which could be used by a company to determine the identity of a user each time he tries to access company resources.

Hence, it is worth considering the efficiency of ZK protocols in a scenario where the prover and the verifier need to run the ZK proof of a statement many times (sequentially). In practice, the state information derived from previous sessions could be reused in the following sessions to achieve significant savings in processing load and bandwidth, e.g., session resumption of TLS 1.3 [55]. It is desired that the ZK protocol for the subsequent sessions be much more efficient than that of the original one. Therefore, a natural question is that

*How can we resume a session of ZK proofs efficiently?*

An intuitive way is to reuse the information of previous ZK sessions (of the same statement). In fact, similar issues have been considered in the research of NIZK. A series of works explored how to reuse the common reference string (CRS) of NIZK for multiple theorems and multiple provers [14,35,46], which implies the case of CRS reuse in multiple sessions (or executions). On the interactive ZK protocols, how to securely reuse previous transcripts among different sessions is, however, more subtle and tends to result in a breach of security. For instance, the witness in many $\Sigma$ protocols can be extracted when the same commitments are reused in different sessions (with different challenges).

In another recent line of works, researchers have shown how to use secure multiparty computation (MPC) to obtain (NI)ZK proofs, and further quantum-resistant signatures. Ishai et al. [48] showed how to use the so-called "MPC-in-the-head" approach to obtain public-coin ZK proofs. Their scheme was further improved by [39,23] to obtain quantum-resistant signature via Fiat-Shamir transformation [36]. The resulting signature Picnic [21], which was submitted to the NIST post-quantum standardization effort, is very competitive, since its security is based entirely on symmetric-key primitives. But it is still less efficient than lattice-based CRYSTALS-DILITHIUM [6] and multivariate-based Rainbow [29]. So we would like to ask whether we could reduce the overall complexity of Picnic when considering multiple sequential signing requests. In other words, *how can we resume the signing/verifying procedure of Picnic efficiently?*

## 1.1 Our Contributions

We introduce the notion of *resumable honest verifier ZK proof of knowledge* (resumable HVZKPoK) to capture the security and efficiency of HVZKPoK in the scenario of session resumption. In that scenario, the prover and the verifier can perform the ZK proofs multiple times sequentially, where each proof is executed in a session. Informally, we say an HVZKPoK is resumable if it satisfies (1) *resumable zero-knowledge*, i.e., no additional information about the witness is revealed from *all* sessions; (2) *resumable knowledge soundness*, i.e., the witness can be extracted from *every* session; (3) *resumption efficiency*, i.e., the cost of the resumed session in terms of both computation and communication should be much less than that of the initial session (or the original ZK proofs).

The main challenge of constructing resumable HVZKPoK is to achieve resumable knowledge soundness while preserving resumption efficiency, since removing or reusing partial transcripts of the original ZK proofs would undermine or break its soundness property in general. To overcome that problem, we investigate the "MPC-in-the-head" paradigm in the preprocessing model proposed by Katz, Kolesnikov and Wang (KKW) [51], and find that their proofs can be separated according to the decomposition of circuits, where the proofs for the corresponding partial circuits can be further rerandomized without breaking the security. By making use of such separability of the KKW proofs, we provide a general construction of the resumable HVZKPoK. The main idea is that the underlying circuits are decomposed into two parts, where the proofs for the partial circuits with smaller size can be rerandomized. Once the initial session of proofs for the entire circuits is finished, both the prover and the verifier can resume a session by running the rerandomized proofs for the partial circuits only. Since the cost of the KKW proofs is closely related to the number of the AND gates of the circuits, the cost of the resumed session is reduced significantly due to the size of the partial circuits.

Notice that only proofs for the partial circuits usually cannot achieve knowledge soundness implied by the proofs for the entire circuits. To mitigate that problem, we identify a property called extractable decomposition, which guarantees that the witness can be extracted from the inputs of the separated partial circuits. Interestingly, most block ciphers satisfy this property. In addition, the circuits of block ciphers can be decomposed such that the separated partial circuits have no AND gates. Hence, the cost for resuming sessions can be made very small when implemented with block ciphers. By applying the Fiat-Shamir heuristic [36], our resumable HVZKPoK can be transformed into a stateful post-quantum signature scheme. Comparing with the typical chain-based stateful signature [50], the main advantage of our scheme is that, once the initial signature has been generated, the subsequent signatures are much more efficient than the initial one.

We implement our signature and give a comparison with Picnic3 [49]. The sign/verify time of our subsequent signatures can be reduced to 3.1%/3.3%-9.2%/8.8% of Picnic3 and the corresponding signature size can be reduced to 36.0%-38.1%. (For the fixed verifier, the size of the state information needed to be stored is about 2.9 KB-10.9KB.) Although the complexity of our first signature is slightly higher than Picnic3, it is worthy for the reducing cost of subsequent signatures. In particular, our stateful signatures make the symmetric-based signatures, such as Picnic3, suitable for the post-quantum secure blockchain protocol. That is, the previous signatures (or the states) can be stored in the history blocks efficiently and publicly. The verifier only needs to check the validity of the current signature without checking all previous signatures, since the validity of the previous signatures is implied by the consistency of the underlying consensus protocol.

Moreover, applying our method to the Cramer, Damgård and Schoenmakers (CDS) technique [26], we construct a compressed one-out-of $N$ proof, where most of the transcripts for the simulation in CDS technique can be removed. Furthermore, we can construct a ring signature from symmetric key primitives using our compressed one-out-of $N$ proof (without resorting to the Merkle-tree based accumulator). Comparing with the ring signatures from symmetric key primitives proposed by [51], the size of our ring signature is about 1/3 of [51] when the ring size is less than $2^4$.

## 1.2 Main Techniques

The intuition behind our construction is that, once the verifier accepts in a session, he is convinced not only that the prover know the witness $w$ in this session, but also the correctness and randomness of the corresponding transcript. Although those transcripts cannot be used as proof for the subsequent sessions directly, the trust implied by those transcripts has been established and can be recycled in the resumed session. More precisely, the proofs for the relation $R$ generated in previous sessions can be interpreted as "certified" commitments for the current session. Then the prover can prove that he knows the witness $w$ for a new relation $R'$ and its consistency with the "certified" commitments, and thus the relation $R$. Both the prover and the verifier can take advantage of the state information of previous sessions to run the ZK proof for $R'$, the proof size of which is required to be shorter than that of $R$.

For example, consider the session resumption of the HVZKPoK for the relation $R$, where $((F, y), w) \in R$ iff $F(w) = y$. A typical $\Sigma$ protocol for $R$ consists of $(a, e, z)$, which denotes commitment, challenge and response, respectively. Let $\pi^{(0)} = (a^{(0)}, e^{(0)}, z^{(0)})$ be the view or the proof generated by the $\Sigma$ protocol during the initial session. Suppose that $F(w)$ can be considered as the composition of $f$ and $g$, i.e., $F(w) = f \circ g(w) = f(w') = y$, where $g(w) = w'$. The related statement can be interpreted as that the prover knows $w$ such that $g(w) = w' \wedge f(w') = y$, where both $w$ and $w'$ are kept secret. We require that $w'$ is "equivalent" to $w$, which means there exits an efficient extractor $\mathcal{E}$ such that $\mathcal{E}(w') = w$. We say the decomposition of $F$ is *extractable* if the above property on $w'$ holds. Next, we can reduce the proof size of the subsequent session by providing the proofs of the statement $f(w') = y$ and its consistency with $F(w) = y$, which can be seen as a new relation $R'$. The proof of consistency with $F(w) = y$ needs to rely on $\pi^0$, which plays the role of parts of the commitments in the following resumed session. Let $\pi^{(0)}||a^{(1)}$ denote the commitment for the proof of $R'$ in the second session. The corresponding challenge and response are denoted as $e^{(1)}$ and $z^{(1)}$, respectively. Suppose $\pi^{(0)}$ is stored in the local storage of the prover and the verifier for a long time. We hope that the proof size of $R'$, say $\pi^{(1)} = (a^{(1)}, z^{(1)}, e^{(1)})$, can be shorter than that of $\pi^{(0)}$. However, there are two problems with the above intuitive construction.

1. Does the reuse of $\pi^{(0)}$ cause any leakage of $w$?
2. How can we find the extractable decomposition of a function $F$?

To solve the first problem, we use $\pi^{(0)}$ as a commitment which will not be "opened" and rerandomize the corresponding parts of $\pi^{(0)}$ when considering the proof of $f(w') = y$. Due to the special property of the KKW proof [51], $\pi^{(0)}$ can be separated according to the decomposition of $F$, and the proof for $f$ can be re-randomized to protect the secret $w$. In particular, the KKW proof [51] considers a masked variant of $F(w) = y$, i.e., $F(\hat{w}) \to \hat{y}$, where $\hat{w} = w \oplus \lambda_w$ and $\hat{y} = y \oplus \lambda_y$ and $(\lambda_w, \lambda_y)$ are random masks, and proves the correctness and consistency of $(w, y, \lambda_w, \lambda_y)$ using the MPC-in-the-head with preprocessing paradigm. Here, we decompose $F(\hat{w}) \to \hat{y}$ as $g(\hat{w}) \to \hat{w}' \wedge f(\hat{w}') \to \hat{y}$, where the variables with hat or bar denote the corresponding masked variables. In the resumed session, $\pi^{(1)}$ actually implies the proof for $g(\hat{w}) \to \hat{w}' \wedge f(\bar{w}') \to \bar{y}$, where $f(\bar{w}') \to \bar{y}$ is the re-randomization of $f(\hat{w}') \to \hat{y}$ with different masks. The proof for $g(\hat{w}) \to \hat{w}'$ is implied by $\pi^{(0)}$, which is interpreted as parts of the commitment in the current session and will not be "opened". Due to the re-randomized masked witness $\bar{w}'$, the zero-knowledge property of the KKW proof for $f(\bar{w}') \to \bar{y}$ and its consistency with $g(\hat{w}) = \hat{w}'$ can be guaranteed.

For the second problem, most block ciphers satisfy the property of extractable decomposition. Specifically, the iterated structure and the key schedule guarantee that $w$ can be extracted from the input $w'$ of the last round, which makes block cipher circuits suitable for our resumable HVZKPoK.

## 1.3 Related Works

**Zero-knowledge from symmetric primitives**. Most efficient ZK proofs exist for a restricted set of languages, e.g., languages relying on algebraic structures.

To construct efficient ZK proofs for a larger class of languages, many works focus on ZK proofs for arbitrary circuits [44,11,38,53,13,25,16,45,19,59,10,12], which have relatively short proofs size and verification time. However, most efficient constructions require a trusted setup or rely on assumptions, which are insecure in the quantum setting.

Ishai et al. [48] introduced a novel way of constructing ZK proofs, called "MPC-in-the-head", which is based on secure multi-party computation (MPC) protocols. Following the idea of [48], Giacomelli et al. [39] proposed ZKBoo which supports efficient non-interactive (NI) ZKPoKs for arbitrary circuits. Chase et al. [23] improved the performance of ZKBoo and proposed ZKB++, which is used to construct the post-quantum secure signature scheme, called Picnic. Compared with other post-quantum secure signature candidates, Picnic relies on the security of the underlying symmetric-key primitives instead of structured hardness assumptions. Although Picnic has good performance on the speed when implemented on hardware, it has a large signature size which is linear in the size of the circuits. Ames et al. [3] proposed Ligero with sublinear proof size, which asymptotically outperforms ZKBoo and ZKB++. Katz et al. [51] instantiated the MPC-in-the-head paradigm in the preprocessing model, which can reduce the number of parallel repetitions, and provided an improved version of Picnic, called Picnic2. Guilhem et al. [57] applied the "MPC-in-the-head with preprocessing" approach to the arithmetic circuit of AES and implemented a signature scheme, called BBQ, whose security is based on AES. Baum et al. [8] proposed a novel way to construct an AES-based signature scheme, called Banquet, which reduces the signature size compared with BBQ and its implementation results show that Banquet can be made almost as efficient as Picnic2 . Baum and Nof [7] incorporated the "sacrificing" paradigm into "MPC-in-the-head" to reduce the proof size for arithmetic circuits. Kales and Zaverucha [49] made further optimizations and presented a new parameter set for Picnic2, called Picnic3. Goel et al. [40] introduced a general framework for constructing $\Sigma$-protocols of disjunctive proof which can be used to implement ring signature. Recently, Goel et al. [41] proposed a novel technique for efficiently adding set membership proofs to any MPC-in-the-head based ZK protocol and the resulting ring signatures outperform Katz et al.'s construction by a factor of 5 to 8.

**Resettable zero knowledge (rZK).** rZK [20,27] can remain zero knowledge even if an adversary can interact with the prover many times, each time resetting the prover to any previous stage. Although rZK is a stronger notion than concurrent zero knowledge, the construction of efficient rZK is rather difficult in practice. Our resumable ZK is reminiscent of rZK in that the prover is rewinded to a previous stage, but we investigate how to rewind the prover in a safe way so that the proofs for subsequent sessions can be more efficient. In particular, our resumable ZK needs to rewind prover to a certain point, where the corresponding data can be rerandomized to ensure the security.

**Proof-carrying data (PCD).** PCD [24,18], which is a generalization of incrementally-verifiable computation (IVC) [58], allows every intermediate state of the distributed computation performed by mutually distrustful parties can be succinctly verified. Our resumable HVZKPoK focuses on the scenario where both the verifier and the statement is fixed. In that scenario, we can recycle the trust established in the initial session, and the proofs for subsequent sessions with better concrete efficiency is possible.

## 2 Preliminaries

**Notations**. Let $[n]$ denote $\{1, \dots, n\}$ and $\kappa$ denote the security parameters. Let $C$ and $C'$ be the boolean circuit representation of $F$ and $f$, respectively, where $C$ and $C'$ consists of XOR and AND gates. Let $|C|$ denotes the number of AND gates in the circuit $C$ and $|C_{in/out}|$ denotes the number of input/output wires of $C$. Let $L_R \subseteq \{0, 1\}^*$ be an NP language and $R$ be the related NP-relation for circuit $C$. $A \approx_c B$ denotes computational indistinguishability between distributions $A$ and $B$. Let $\mathsf{Com}$ denote a commitment scheme. A commitment to a message $m$ is denoted as $\mathsf{com} = \mathsf{Com}(m; r)$ where $r \in \{0, 1\}^\kappa$ is chosen uniformly. We say $\mathsf{Com}$ is secure if it satisfies the following properties: (1) *Hiding*: $\mathsf{Com}(m; r)$ reveals noting about $m$; (2) *Binding*: it is hard to find two messages $m \neq m'$ such that $\mathsf{Com}(m; r) = \mathsf{Com}(m'; r')$. Let $H$ denote the hash function. We say $H$ is collision-resistant if the probability that any PPT adversary finds $x$ and $x'$ such that $H(x) = H(x')$ and $x \neq x'$ is negligible.

### 2.1 MPC-in-the-head with preprocessing

MPC-in-the-head paradigm [48] is a novel technique to construct ZK proofs from MPC protocols. Suppose the statement to be proven is $(C, y)$, where $C(w) = y$ and $w$ is the witness. Following the MPC-in-the-head paradigm, the prover simulates an MPC protocol which evaluates the circuit $C$ among all the parties *in his head* and the input of each party is a secret sharing of the witness $w$. The prover then commits to the views of each party in the execution of the MPC protocol. The verifier randomly chooses a subset of these commitments as the challenge. Once receiving the challenge, the prover opens the challenged commitments. The verifier checks the correctness and consistency of these views.

MPC-in-the-head with preprocessing (KKW) [51] improves MPC-in-the-head paradigm and the resulting scheme can achieve the required soundness with much shorter proofs. Loosely speaking, the KKW protocol has two phases, the *preprocessing* phase and the *online* phase. In the *preprocessing* phase, the prover generates random masks for each party, which are used to hide the witness. In the *online* phase, the prover simulates the execution of the MPC protocol using the masked shares of each party and the masked input (or the masked witness) of the circuit. Note that the verifier needs to challenge both phases. The main techniques of MPC-in-the-head with preprocessing are described below.

Let $[x]$ denote an $n$-out-of-$n$ (XOR-based) secret sharing scheme of a bit $x$, i.e., $x = [x]_1 \oplus \cdots \oplus [x]_n$, where $[x]_i$ for $1 \leq i \leq n$ is the secret share. Suppose the underlying $n$-party MPC protocol is $\Pi$, which is executed by $n$ parties $P_1, \cdots, P_n$. Let $z_\alpha$ denote the value of wire $\alpha$ of $C(w)$. $z_\alpha$ will be masked by a random bit $\lambda_\alpha$, say, $\hat{z}_\alpha = z_\alpha \oplus \lambda_\alpha$. Each party $P_i$ will hold $[\lambda_\alpha]_i$, which is a share of $\lambda_\alpha$.

- **Preprocessing phase**. In the preprocessing phase, the prover generates the masks for each party $P_i$. More precisely, $P_i$ is given the following values.
  - $[\lambda_\alpha]_i$ for each input wire $\alpha$.
  - $[\lambda_\gamma]_i$ for the output wire $\gamma$ of each AND gate.
  - $[\lambda_{\alpha,\beta}]_i$ for each AND gate with input wires $\alpha$ and $\beta$ such that $\lambda_{\alpha,\beta} = \lambda_\alpha \cdot \lambda_\beta$.

  $[\lambda_\alpha]_i$ and $[\lambda_\gamma]_i$ can be generated using a pseudorandom generator (PRG) with a random seed $\mathsf{seed}_i$, for $i = 1, \ldots, n$, where $[\lambda_\alpha]_1 \oplus \cdots \oplus [\lambda_\alpha]_n = \lambda_\alpha$ and $[\lambda_\gamma]_1 \oplus \cdots \oplus [\lambda_\gamma]_n = \lambda_\gamma$. Hence, $\mathsf{seed}_i$ instead of $\{[\lambda_\alpha]_i\}$ and $\{[\lambda_\gamma]_i\}$ is given to $P_i$ so that the total proof size can be reduced. Notice that $[\lambda_{\alpha,\beta}]_n$ cannot be generated using $\mathsf{seed}_n$ only due to $\lambda_{\alpha,\beta} = \lambda_\alpha \cdot \lambda_\beta$. Actually, $n-1$ shares of $\lambda_{\alpha,\beta}$ are generated using PRG, while the share of $P_n$ is computed by $[\lambda_{\alpha,\beta}]_n := \lambda_\alpha\lambda_\beta \oplus [\lambda_{\alpha,\beta}]_1 \oplus \cdots [\lambda_{\alpha,\beta}]_{n-1}$, which plays the role of "correction bits". Therefore, party $P_n$ needs to be given $\mathsf{aux}_n = \{[\lambda_{\alpha,\beta}]_n\}$ for all AND gates in addition to $\mathsf{seed}_n$.
- **Online phase**. During the online phase, each party $P_i$ runs the underlying $n$-party MPC protocol $\Pi$ to evaluate the circuit $C$ gate-by-gate in topological order. For each gate with input wires $\alpha$ and $\beta$ and output wire $\gamma$,
  - For an XOR gate, $P_i$ can locally compute $\hat{z}_\gamma = \hat{z}_\alpha \oplus \hat{z}_\beta$ and $[\lambda_\gamma]_i = [\lambda_\alpha]_i \oplus [\lambda_\beta]_i$, since $P_i$ already holds $\hat{z}_\alpha$, $[\lambda_\alpha]_i$, $\hat{z}_\beta$ and $[\lambda_\beta]_i$.
  - For an AND gate, $P_i$ locally computes $[s]_i = \hat{z}_\alpha[\lambda_\beta]_i \oplus \hat{z}_\beta[\lambda_\alpha]_i \oplus [\lambda_{\alpha,\beta}]_i \oplus [\lambda_\gamma]_i$, publicly reconstructs $s$, and computes $\hat{z}_\gamma = s \oplus \hat{z}_\alpha\hat{z}_\beta$ which satisfies $\hat{z}_\gamma = z_\gamma \oplus \lambda_\gamma$. Note that party $P_i$ holds $[\lambda_{\alpha,\beta}]_i$ and $[\lambda_\gamma]_i$ in addition to $\hat{z}_\alpha$, $[\lambda_\alpha]_i$, $\hat{z}_\beta$ and $[\lambda_\beta]_i$ for each AND gate.

  Finally, each party $P_i$ can compute $\hat{z}_\gamma$ for the output wire $\gamma$ of the circuit, and the output value $z_\gamma$ is computed as $z_\gamma = \hat{z}_\gamma \oplus \lambda_\gamma$, where $\lambda_\gamma$ is reconstructed publicly.

**Security of the underlying MPC protocol**. [22, Lemma 6.1] proves that the underlying MPC protocol $\Pi_{mpc}$ is secure against an all-but-one corruption in the semi-honest model by showing that there exists a simulator for the MPC protocol $\Pi_{mpc}$ such that the real execution of $\Pi_{mpc}$ is computational indistinguishable from the simulated execution of $\Pi_{mpc}$ under the assumption of secure PRG.

## 3 Resumable HVZK Proof of Knowledge

Let $R$ be an efficiently decidable binary NP-relation which is polynomially bounded, and $L_R$ be the NP-language defined by $R$. That is, $\exists w$ such that $(x, w) \in R$ iff $x \in L_R$. In our setting, the prover and the verifier can sequentially perform the zero-knowledge proofs for $L_R$ polynomially-many times, say $q(\kappa)$ times, where each proof is modeled as a session and the $t$-th session is denoted as $\mathsf{session}(t)$, for $t \in \{1, \ldots, q(\kappa)\}$. In each $\mathsf{session}(t)$, the prover aims to convince the verifier that he knows the witness $w$ for statement $x$ by running the

HVZKPoK protocol $\mathit{\Pi} = \{(\mathcal{P}^{(t)}, \mathcal{V}^{(t)})\}$. Let $\mathcal{P}^{(t)} = \mathcal{P}(x, w, pr_t, ps_t)$ denote the prover's strategy of $\mathsf{session}(t)$, which takes as input the common-input $x$, witness $w$, prover's randomness $pr_t$ and state $ps_t$. Here, $ps_t$ is the prover's state after $\mathsf{session}(t-1)$. Similarly, let $\mathcal{V}^{(t)} = \mathcal{V}(x, vr_t, vs_t)$ denote the verifier's strategy of $\mathsf{session}(t)$, which takes as inputs the common-input $x$, verifier's randomness $vr_t$ and state $vs_t$.

In this paper, we transform an "ordinary" HVZKPoK $\mathit{\Pi}' = (\mathcal{P}', \mathcal{V}')$ to a resumable HVZKPoK $\mathit{\Pi} = \{(\mathcal{P}^{(t)}, \mathcal{V}^{(t)})\}$, the security of which is more subtle. In particular, we have to ensure that the adversary who does not have the knowledge of the witness cannot convince the verifier in any session, even that the adversary can have access to the transcripts of all previous sessions. Consider the following game on soundness. The adversary $A$ can invoke the "honest" prover[6] to run $\mathit{\Pi}$ for $x \in L_R$ with the verifier for polynomially-many sequential sessions, say, $\mathsf{session}(1), \ldots,\ \mathsf{session}(q(\kappa) - 1)$, where $A$ can get all the transcripts of these sessions. For the next session, say $\mathsf{session}\ q(\kappa)$, $A$ runs $\mathit{\Pi}$ with the verifier for $x \in L_R$, trying to convince the verifier without the help of the "honest" prover. The soundness of the resumable HVZK is defined according the above game, which requires that $A$ can win the game only with negligible probability. Formal definition of resumable HVZKPoK is described below.[7]

**Definition 1 (Resumable HVZK Proof of Knowledge).** $\mathit{\Pi} = \{(\mathcal{P}^{(t)}, \mathcal{V}^{(t)})\}$ *is a resumable honest verifier zero-knowledge proof of knowledge for the relation* $R$ *with soundness error* $\xi$ *if the following properties hold:*

- **Completeness:** *If the prover and the verifier follow the protocol* $(\mathcal{P}^{(t)}, \mathcal{V}^{(t)})$ *on inputs* $x \in L_R$ *and witness* $w \in R_x$*, then the verifier always accepts in each* $\mathsf{session}(t)$*, for* $t \in \{1, \ldots, q(\kappa)\}$.
- **Resumable Honest Verifier Zero-Knowledge:** *Let* $\mathsf{view}_{\mathcal{V}}^{\mathcal{P}}(x, w)$ *be the transcripts of all the sessions run by the prover and the verifier. There exists a PPT simulator* $\mathsf{Sim}$ *such that* $\mathsf{Sim}(x) \approx_c \mathsf{view}_{\mathcal{V}}^{\mathcal{P}}(x, w)$ *for all* $x \in L_R$ *and* $w \in R_x$.
- **Resumable Knowledge Soundness:** *For each* $\mathsf{session}(t)$*, there exists a probabilistic knowledge extractor* $\mathcal{E}$*, such that for every* $\hat{\mathcal{P}}^{(t)}$ *and every* $x \in L_R$*, the algorithm* $\mathcal{E}$ *satisfies the following condition:*
  - *Let* $\delta_t(x)$ *be the probability that the verifier accepts on input* $x$ *for* $(\hat{\mathcal{P}}^{(t)}, \mathcal{V}^{(t)})$ *of* $\mathsf{session}(t)$*. If* $\delta_t(x) > \xi_t(x)$*, then upon input* $x \in L_R$ *and oracle access to* $\hat{\mathcal{P}}^{(t)}$*, the algorithm* $\mathcal{E}$ *outputs a valid witness* $w \in R_x$ *in expected number of steps bounded by* $O(\frac{1}{\delta_t(x) - \xi_t(x)})$.

[6] In our setting, a prover who has the knowledge of the witness is considered to be honest, which means he follows the protocol honestly. We do not consider the case that a prover who has the knowledge of the witness tries to convince the verifier in a resumed session by deviating the protocol.

[7] The concurrent zero-knowledge [34] seems to be a stronger notion than ours when considering zero-knowledge and soundness only. But the requirement on resumption efficiency cannot be implied by the concurrent zero-knowledge. Although resumable HVZKPoK in the concurrent setting is beyond the scope of this paper, we believe it is another interesting topic.

*Here, $\xi_t$ denotes the soundness error of* $\mathsf{session}(t)$ *for* $t \in \{1, \ldots, q(\kappa)\}$*. Let* $\xi = \max\{\xi_1, \ldots, \xi_{q(\kappa)}\}$*.*

– **Resumption efficiency:** *For each* $\mathsf{session}(t)$ *with* $t > 1$*, the computational and communicational complexity of* $(\mathcal{P}^{(t)}, \mathcal{V}^{(t)})$ *should be less than that of the original HVZKPoK* $\Pi' = (\mathcal{P}', \mathcal{V}')$ *for* $R$*. That is, resumable HVZK proof of knowledge should be efficient in each resumed session.*

Here, the statement $x$ of $R$ is of the form $(F, y)$, such that $(x, w) \in R$ iff $F(w) = y$, where $F$ denotes a function. Let $C$ be the circuit representation of $F$. So the statement can be rewritten as $(C, y)$, such that $(x, w) \in R$ iff $C(w) = y$. As mentioned in section 1, the function $F$ needs to satisfy a special property called extractable decomposition, which is defined as follows.

**Definition 2 (Extractable Decomposition).** *Let* $F : \{0,1\}^\kappa \to \{0,1\}^{\kappa'}$ *be a function which has a decomposition as* $F = f \circ g$*. We say the decomposition is extractable if, for all* $x \in \{0,1\}^\kappa$*, there exists an efficient extractor* $\mathcal{E}_D$ *such that* $\mathcal{E}_D(g(x)) = x$*.*

Consider the case that $F(w) = \mathsf{Enc}(w, m)$, where $\mathsf{Enc}(w, m)$ is a block cipher with the private key $w$ and the plaintext $m$. It is known that a typical block cipher consists of multiple rounds, where each round takes as inputs the output of previous round and the corresponding subkey (or round key) derived from the *master* key $w$. Note that the subkey schedule of most block ciphers is reversable, which implies most block ciphers naturally satisfy the property of extractable decomposition. Concretely, given a block cipher with $n$ rounds, the first $n-1$ rounds as well as the key schedule can be taken as $g$, and the last round is taken as $f$. Suppose the output of $g(w)$ is $w'$, which consists of the output of the $(n-1)$-th round and $n$-th round key $k_n$. $f$ takes as input $w'$ and outputs the final ciphertext. Obviously, $w$ can be extracted from $w'$, which implies the extractability of the decomposition[8].

## 4 General Construction for Resumable HVZKPoK

In this section, we present the general construction of resumable HVZKPoK from the KKW protocol. We first abstract the construction of the original KKW protocol [51] for $F(w) = y$, where $F$ has a decomposition as $F = f \circ g$. Then, we show how to efficiently resume HVZKPoK for $w'$ such that $f(w') = y$ and $w' = g(w)$. Recall that $C$ and $C'$ denote the circuit representation of $F$ and $f$, respectively. So $F(w) = y$ and $f(w') = y$ can be rewritten as $C(w) = y$ and $C'(w') = y$, respectively.

[8] Consider a special case that $F = f \circ g(\omega)$, where $g(\omega) = (C'(\omega), \omega)$ for some circuit $C'$ and $f$ just outputs one bit of $g(\omega)$. Our construction for resumable HVZKPoK is still suitable for such decomposition.

### 4.1 KKW Protocol for $F$

The KKW protocol $\pi^F$ for $C(w) = y$, i.e., $F(w) = y$, consists of the preprocessing phase $\pi^F_{pre}$ and the online phase $\pi^F_{on}$. $\pi^F_{pre}$ shows that the $n$ parties' states are generated randomly and correctly by "cut-and-choose", and $\pi^F_{on}$ ensures that each party's view in the MPC protocol are correct and consistent. Let $M$ denote the number of repetitions for reducing the soundness error.
**Preprocessing phase** $\pi^F_{pre}(1^\kappa)$.

- Round 1. *Commit* to the masks of $M$ instances.
  The prover prepares the masks $\lambda_j$ of the MPC protocol for the circuit $C$ as described in section 2.1 for each instance $j \in [M]$. Since $\lambda_j$ is determined by $n$ parties' states $\{\mathsf{state}_{j,1}, \ldots, \mathsf{state}_{j,n}\}$, which are the random seeds and the $n$-th party's auxiliary information, the prover only needs to commit to those states. The corresponding commitments are denoted as $\mathsf{com}^F_{pre}$. The prover sends $\mathsf{com}^F_{pre}$ to the verifier.
- Round 2. *Challenge* for the preprocessing phase.
  The verifier chooses a random subset $\mathcal{C} \in [M]$ with $|\mathcal{C}| = \tau$, which is used to challenge the prover to open the commitments of instances in $[M]\backslash\mathcal{C}$, so that the verifier can check the randomness and correctness of $\lambda_j$ for each instance $j \in [M]\backslash\mathcal{C}$. The verifier sends $\mathcal{C}$ to the prover.
- Round 3-a. *Respond* to the challenge for the preprocessing phase.
  The prover computes the openings of the commitments of instances in $[M]\backslash\mathcal{C}$. Denote these openings as $\mathsf{resp}^F_{pre}$. The prover sends $\mathsf{resp}^F_{pre}$ to the verifier.

**Online phase** $\pi^F_{on}(w, \{\mathsf{state}_{j,i}\}_{j\in\mathcal{C}, i\in[n]})$.

- Round 3-b. *Commit* to the views of each party.
  The prover runs the $n$-party MPC protocol for $C(w) = y$ using the masks $\lambda_j$ and the witness $w$ for each instance $j \in \mathcal{C}$, and computes the commitments to the views of each party in the MPC protocol execution. Let $\mathsf{com}^F_{on}$ denote these commitments. (For simplicity, the masked values of input, e.g., $\hat{w}_j = w \oplus \lambda_{j,w}$, is considered to be part of $\mathsf{com}^F_{on}$.) The prover sends $\mathsf{com}^F_{on}$ to the verifier.
- Round 4. *Challenge* for the online phase.
  The verifier chooses a random set $\mathcal{P} = \{p_j\}_{j\in\mathcal{C}}$ with $p_j \in [n]$, which is used to challenge the prover to open the views of all but the $p_j$-th party for each instance $j \in \mathcal{C}$, so that the verifier can check the consistency of $n-1$ parties' views for that instance. The verifier sends $\mathcal{P}$ to the prover.
- Round 5. *Respond* to the challenge for the online phase.
  The prover computes the openings of all but the $p_j$-th party's commitments for each instance $j \in \mathcal{C}$. Let $\mathsf{resp}^F_{on}$ denote these openings. The prover sends $\mathsf{resp}^F_{on}$ to the verifier.

**Verification Strategy**

1. For the opened instances in $[M]\backslash\mathcal{C}$, the verifier uses $\mathsf{resp}^F_{pre}$ to recover parts of the openings of $\mathsf{com}^F_{pre}$, which are also used to check the randomness and correctness of the masks.

2. For each unopened instance in $\mathcal{C}$, the verifier uses $\mathsf{resp}_{on}^F$ and the masked values of input to simulate the MPC protocol for $C(w) = y$ and recover the openings of $\mathsf{com}_{on}^F$ and the remaining openings of $\mathsf{com}_{pre}^F$.
3. The verifier checks the output of the simulation of the MPC protocol and the consistency of $\mathsf{com}_{pre}^F$ and $\mathsf{com}_{on}^F$.

## 4.2 Intuitive Construction for Resumable HVZKPoK

Once the verifier accepts the proof, he is convinced not only that the prover has the witness in the current session, but also the correctness and randomness of the transcripts implied by the proof. We notice that the verifier's trust on some transcripts of KKW protocol can be "reused" to reduce the cost of proofs when resuming sessions. An intuitive construction of resumable HVZKPoK for $F$ is described as follows, where the decomposition $F = f \circ g$ is public. For simplicity, we only consider the case of two sessions.

– **HVZKPoK for the initial session.** It is similar to the original KKW protocol, except that the prover needs to prepare preprocessing values for the next session and proves the consistency of $w'$ and $w$. Let $\pi^f = (\pi_{pre}^f, \pi_{on}^f)$ denote the KKW proof for $C'(w') = y$, i.e., $f(w') = y$. HVZKPoK for the initial session consists of the following phases.
  1. $\pi^F$: The original KKW proof for $w$ such that $C(w) = y$.
  2. $\pi_{pre}^f$: Prepare the preprocessing values of $C'(w') = y$ for the next session. In particular, $\pi^F$ and $\pi_{pre}^f$ can be merged with the same preprocessing challenge $\mathcal{C}$, which will be explained later.
  3. $\pi_{cert}$: Consistency proof for $w$ and $w'$. That is, we need to guarantee that $w'$ used in the next session is the correct intermediate value of $C(\cdot)$ when evaluating on input $w$.
– **HVZKPoK for the second session.**
  1. $\pi_{on}^f$: Online phase of the KKW proof for $C'(w') = y$.

Note that $\pi_{pre}^f$ and $\pi_{on}^f$ constitutes the complete KKW protocol for $C'(w') = y$. Intuitively, if the verifier can be convinced that the preprocessing data in $\pi_{pre}^f$ is generated correctly, the prover only needs to run $\pi_{on}^f$ for the second session. Suppose the verifier has accepted the initial session. Combining with the consistency proof $\pi_{cert}$ for $w$ and $w'$, the verifier can be convinced that the prover has the knowledge of $w$ in the second session. Therefore, the prover needs to provide efficient consistency proof for $w$ and $w'$, while ensuring that the preprocessing data in $\pi_{pre}^f$ are generated by the "honest" prover, who has the knowledge of $w$. (Recall the soundness game mentioned in section 3, where we do not consider the malicious prover who has the knowledge of witness.) To do so, we modify the KKW protocol $\pi_{pre}^f$ for $C'(w') = y$.

## 4.3 Modified KKW for $f$ and Consistency Proof

Suppose $\lambda_{w'}$ is the random masks for $w'$ in $\pi^F$, i.e., the masked intermediate value $\hat{w}' = w' \oplus \lambda_{w'}$. The main modification of $\pi^f$ is that the generation of the preprocessing data in $\pi^f_{pre}$ is based on $\lambda_{w'}$. More specifically, the prover rerandomizes $\hat{w}'$ using a random and public value $\Delta$, i.e., $\bar{w}' = w' \oplus \lambda_{w'} \oplus \Delta$. Then, the prover generates the corresponding preprocessing values for the KKW proof for $C'(w') = y$ using $\lambda'_{w'} = \lambda_{w'} \oplus \Delta$ as the mask. Here, the prover generates $n-1$ secret shares of $\lambda'_{w'}$ by running PRG with $n-1$ random seeds, while the $n$-th secret share $[\lambda'_{w'}]_n$ is determined by $[\lambda'_{w'}]_n = \lambda'_{w'} \oplus [\lambda'_{w'}]_1 \oplus \cdots \oplus [\lambda'_{w'}]_{n-1}$ and is sent to the verifier. ($\mathsf{com}^f_{pre}$ commits to the corresponding seeds for each party.) Hence, the verifier only needs to challenge the prover to open $n-2$ parties' views in the online phase.

Based on the above modification, we can provide a simple and efficient construction for the consistency proof $\pi_{cert}$. After $\pi^f_{pre}$, the prover computes a commitment $\mathsf{com}_{on}$, which commits to $\mathsf{com}^F_{on}||\mathsf{com}^f_{pre}||\Delta||[\lambda'_{w'}]_n$, and sends $\mathsf{com}_{on}$ as well as $\mathsf{com}^f_{pre}||\Delta||[\lambda'_{w'}]_n$ to the verifier. Since the verifier accepted the initial session, the consistency of $\mathsf{com}^F_{on}$ and the preprocessing data of the initial session, e.g., $\lambda_{w'}$, has been checked. Due to the binding property of $\mathsf{com}_{on}$, the rerandomized mask $\lambda'_{w'} = \lambda_{w'} \oplus \Delta$, which is determined by $\mathsf{com}^f_{pre}||\Delta$, is hard to be modified. In the second session, by checking the openings of $\mathsf{com}^f_{pre}$, it is implicitly guaranteed that $\lambda'_{w'}$ is generated by the same "honest" prover of the initial session. Therefore, the witness $w'$ implied by $\bar{w}'$ is the same as that of the initial session, i.e., $\bar{w}' = w' \oplus \lambda'_{w'}$, and the verifier does not need to check the correctness and randomness of the preprocessing data for the second session by "cut-and-choose". Notice that only the unopened instances in the initial session need the simulated executions of the MPC protocol, which means only in these instances, we need to rerandomize the masked intermediate value $\hat{w}'$. That is why $\pi^F$ and $\pi^f_{pre}$ can be merged with the same preprocessing challenge $\mathcal{C}$. To summarize, the modified KKW $\pi^f = (\pi^f_{pre}, \pi^f_{on})$ for the partial circuits $C'$ is described as follows.

**Preprocessing phase** $\pi^f_{pre}(\{\lambda_{j,w'}\}_{j\in\mathcal{C}}, \mathcal{C})$.

– Round 1-a. *Rerandomize* the masks for the instances in $\mathcal{C}$.
  1. The prover chooses a random $\mathsf{seed}^\Delta$ to generate $\Delta_j$ for each instance $j \in \mathcal{C}$. Then, the prover computes the rerandomized masks $\lambda'_{j,w'} = \lambda_{j,w'} \oplus \Delta_j$, where $\lambda_{j,w'}$ is the mask of $\hat{w}'_j$.
  2. For each instance $j \in \mathcal{C}$, the prover chooses a random $\mathsf{seed}_{j,i}$ for each party $P_i$ to generate the share $[\lambda'_{j,w'}]_i$ for $i \in [n-1]$, while $[\lambda'_{j,w'}]_n$ is computed by $[\lambda'_{j,w'}]_n = \lambda'_{j,w'} \oplus [\lambda'_{j,w'}]_1 \oplus \cdots \oplus [\lambda'_{j,w'}]_{n-1}$. Other preprocessing values are generated as described in Section 2.1. $[\lambda'_{j,w'}]_n$ is included as part of the auxiliary information $\mathsf{aux}_n$. So we have $\mathsf{aux}_n \in \{0,1\}^{|C'|+|C'_{in}|}$. The prover sets $\mathsf{state}'_{j,i} = \mathsf{seed}_{j,i}$ for $i \in [n-1]$, and $\mathsf{state}'_{j,n} = \mathsf{seed}_{j,n}||\mathsf{aux}_n$. Compute $\mathsf{com}^f_{pre}$, which commits to the $n$ parties' states.
  3. The prover sends $\mathsf{state}'_{j,n}$, $\mathsf{seed}^\Delta$ and $\mathsf{com}^f_{pre}$ to the verifier.

**Online phase** $\pi^f_{on}(w', \{\mathsf{state}'_{j,i}\}_{j\in\mathcal{C}, i\in[n]})$.

– Round 1-b. *Commit* to the views of each party.

The prover runs the MPC protocol for $C'(w') = y$ using the rerandomized masks $\lambda'_{j,w'}$ (determined by $\{\mathsf{state}'_{j,i}\}_{i\in[n]}$) and the witness $w'$ for each instance $j \in \mathcal{C}$. The prover computes the commitment to the views of each party during the MPC protocol. Denote these commitments as $\mathsf{com}^f_{on}$. Send $\mathsf{com}^f_{on}$ to the verifier.

– Round 2. *Challenge* for the online phase.
  The verifier chooses a random set $\mathcal{P} = \{p_j\}_{j\in\mathcal{C}}$ with $p_j \in [n-1]$ in order to challenge the prover to open the views of all but the $p_j$-th party for each instance $j \in \mathcal{C}$, so that the verifier can check the consistency of $n-2$ views for that instance. The verifier sends $\mathcal{P}$ to the prover.
– Round 3. *Respond* to the challenge for the online phase.
  The prover computes the openings of the commitments of those challenged parties for each instance $j \in \mathcal{C}$. Denote the response by $\mathsf{resp}^f_{on}$. Send $\mathsf{resp}^f_{on}$ to the verifier.

**Verification Strategy** The verification strategy is similar to that of the original KKW, except that there is no need to check the randomness and correctness of the masks by cut-and-choose.

1. For each instance $j \in \mathcal{C}$, the verifier uses $\mathsf{resp}^f_{on}$, $\hat{w}'_j$, $\mathsf{seed}^{\Delta}$ and $\mathsf{state}'_{j,n}$ to simulate the MPC protocol for $C'(w') = y$ and recover the openings of $\mathsf{com}^f_{on}$ and $\mathsf{com}^f_{pre}$. (Here, the verifier can get $\hat{w}'_j$ after the initial session.)
2. The verifier checks the output of the simulation of the MPC protocol for $C'$ and the consistency of $\mathsf{com}^f_{on}$ and $\mathsf{com}^f_{pre}$.

## 5 Resumable HVZKPoK from KKW

Using the KKW protocol $\pi^F$ for $C(w) = y$ and the modified version $\pi^f$ for $C'(w') = y$ as building blocks, we can construct the resumable HVZKPoK protocol $\mathit{\Pi}_{Res}$, which consists of two sub protocols $\mathit{\Pi}_{Res,1}$ and $\mathit{\Pi}_{Res,2}$. $\mathit{\Pi}_{Res,1}$ is for the initial session and $\mathit{\Pi}_{Res,2}$ is for the resumed session. Fig. 1 shows the relations of the sub protocols of our general construction.

### 5.1 Resumable HVZKPoK for Initial Session $\boldsymbol{\mathit{\Pi}_{Res,1}}$

The resumable HVZKPoK $\mathit{\Pi}_{Res,1} = (\pi_{pre,1}, \pi_{on,1})$ for the initial session is described as follows.
**Preprocessing phase** $\pi_{pre,1} = \pi^F_{pre}(1^\kappa)$. The preprocessing phase is the same as that of $\pi^F_{pre}(1^\kappa)$.

– Round 1. *Commit* to the masks of $M$ instances.
  The prover runs round 1 of $\pi^F_{pre}(1^\kappa)$, which computes the commitments $\mathsf{com}^F_{pre,1}$ to $\{\mathsf{state}_{j,i,1}\}_{j\in[M],i\in[n]}$ for $M$ instances, and sends $\mathsf{com}^F_{pre,1}$ to the verifier.
– Round 2. *Challenge* for the preprocessing phase.
  The verifier runs round 2 of $\pi^F_{pre}(1^\kappa)$ to send a random $\tau$-sized set $\mathcal{C}$ to the prover.

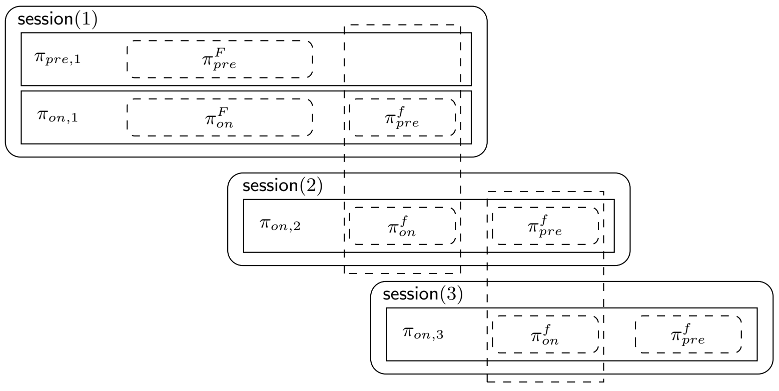

**Fig. 1.** General construction for resumable HVZKPoK

- Round 3-a. *Respond* to the challenge for the preprocessing phase. The prover runs round 3 of $\pi^F_{pre}(1^\kappa)$ to generate the corresponding response, denoted by $\mathsf{resp}^F_{pre,1}$, and sends it to the verifier.

**Online phase** $\pi_{on,1} = (\pi^F_{on}(w, \{\mathsf{state}_{j,i,1}\}_{j\in\mathcal{C},i\in[n]}) \wedge \pi^f_{pre}(\{\lambda_{j,w'}\}_{j\in\mathcal{C}}, \mathcal{C}) \wedge \pi_{cert})$. The prover's strategy of $\pi_{on,1}$ includes: (1) Run $\pi^F_{on}(w, \{\mathsf{state}_{j,i,1}\}_{j\in\mathcal{C},i\in[n]})$ as the original KKW protocol; (2) Prepare the masks for the next session; (3) Certify these masks for the next session to ensure that the witness $w'$ which will be used in the next session is consistent with $w$.

- Round 3-b.
  1. Run round 3-b of $\pi^F_{on}(\omega, \{\mathsf{state}_{j,i,1}\}_{j\in\mathcal{C},i\in[n]})$ to generate the corresponding commitment $\mathsf{com}^F_{on,1}$. Denote the corresponding intermediate mask as $\lambda_{j,w'}$ for each instance $j \in \mathcal{C}$.
  2. Run round 1-a of $\pi^f_{pre}(\{\lambda_{j,w'}\}_{j\in\mathcal{C}}, \mathcal{C})$ to generate a random $\mathsf{seed}^\Delta_2$, the state of each party $\{\mathsf{state}'_{j,i,2}\}_{j\in\mathcal{C},i\in[n]}$ and $\mathsf{com}^f_{pre,2}$ as described above for the next session.
  3. Run $\pi_{cert}$ to commit to $\mathsf{com}^F_{on,1}||\mathsf{com}^f_{pre,2}||\mathsf{seed}^\Delta_2||\{\mathsf{state}'_{j,n,2}\}_{j\in\mathcal{C}}$. Denote the corresponding commitment as $\mathsf{com}_{on,1}$. Send $\mathsf{com}_{on,1}$, $\mathsf{com}^f_{pre,2}$, $\mathsf{seed}^\Delta_2$ and $\{\mathsf{state}'_{j,n,2}\}_{j\in\mathcal{C}}$ to the verifier.
- Round 4. *Challenge* for the online phase. The verifier runs round 4 of $\pi^F_{on}(w, \{\mathsf{state}_{j,i,1}\}_{j\in\mathcal{C},i\in[n]})$ to send a random set $\mathcal{P} = \{p_j\}_{j\in\mathcal{C}}$ with $p_j \in [n]$ to the prover.
- Round 5. *Respond* to the challenge for the online phase. The prover runs round 5 of $\pi^F_{on}(w, \{\mathsf{state}_{j,i,1}\}_{j\in\mathcal{C},i\in[n]})$ to generate the corresponding response $\mathsf{resp}^F_{on,1}$, and sends it to the verifier.

**Verification Strategy**. The verification strategy is similar to that of $\pi^F$, except that the verifier needs to check the consistency of $\mathsf{com}_{on,1}$.

1. For the opened instances in $[M]\backslash\mathcal{C}$, the verifier uses $\mathsf{resp}^F_{pre,1}$ to recover parts of the openings of $\mathsf{com}^F_{pre,1}$, which are also used to check the randomness and correctness of masks.
2. For each unopened instance in $\mathcal{C}$, the verifier uses $\mathsf{resp}^F_{on,1}$ to simulate the MPC protocol for $C(w) = y$ and recover the openings of $\mathsf{com}^F_{on,1}$ and the remaining openings of $\mathsf{com}^F_{pre,1}$.
3. The verifier checks the output of the simulation of the MPC protocol and the consistency of $\mathsf{com}^F_{pre,1}$.
4. The verifier checks the consistency of $\mathsf{com}^F_{on,1}$ and $\mathsf{com}_{on,1}$.

**State Update**. The prover and the verifier need to maintain states for session resumption. The prover's initial state is $\mathsf{pstate} = w$, and the verifier's is $\mathsf{vstate} = \perp$. After the initial session, the prover and the verifier update their states as follows.

– Prover's state update: $\mathsf{pstate} = \{\hat{w}'_j\}_{j\in\mathcal{C}}||\mathsf{seed}^{\Delta}_2||\{\mathsf{state}'_{j,i,2}\}_{j\in\mathcal{C},i\in[n]}$, where $\hat{w}'_j$ is the masked intermediate value for each instance $j \in \mathcal{C}$.
– Verifier's state update: $\mathsf{vstate} = \{\hat{w}'_j\}_{j\in\mathcal{C}}||\mathsf{seed}^{\Delta}_2||\{\mathsf{state}'_{j,n,2}\}_{j\in\mathcal{C}}||\mathsf{com}^f_{pre,2}$.

We emphasize that $\mathsf{vstate}$ can be made public.

## 5.2 Resumable HVZKPoK for Second Session $\Pi_{Res,2}$

For simplicity, we present the resumable HVZKPoK $\Pi_{Res,2}$ for the second session, which can be easily extended to the case of $\mathsf{session}(t)$ for any $t > 1$.
**Online phase** $\pi_{on,2} = (\pi^f_{on}(w', \{\mathsf{state}'_{j,i,2}\}_{j\in\mathcal{C},i\in[n]}) \wedge \pi^f_{pre}(\{\lambda_{j,w'}\}_{j\in\mathcal{C}}, \mathcal{C}) \wedge \pi_{cert})$.

The prover's strategy $\pi_{on,2}$ is similar to $\pi_{on,1}$, except that he simulates the MPC protocol for $C'(w') = y$ instead of $C(w) = y$. Note that all the inputs of $\pi_{on,2}$ can be extracted from $\mathsf{pstate}$.

– Round 1.
  1. Run round 3-b of $\pi^f_{on}(w', \{\mathsf{state}'_{j,i,2}\}_{j\in\mathcal{C},i\in[n]})$ to generate the corresponding commitment $\mathsf{com}^f_{on,2}$.
  2. Run round 1-a of $\pi^f_{pre}(\{\lambda_{j,w'}\}_{j\in\mathcal{C}}, \mathcal{C})$ to generate a random $\mathsf{seed}^{\Delta}_3$, the state of each party $\{\mathsf{state}'_{j,i,3}\}_{j\in\mathcal{C},i\in[n]}$ and $\mathsf{com}^f_{pre,3}$ as described above for the next session.
  3. Run $\pi_{cert}$ to generate $\mathsf{com}_{on,2}$, which is the commitment of $\mathsf{com}^f_{on,2}||\mathsf{com}^f_{pre,3}$ $||\mathsf{seed}^{\Delta}_3||\{\mathsf{state}'_{j,n,3}\}_{j\in\mathcal{C}}$. Send $\mathsf{com}_{on,2}$, $\mathsf{com}^f_{pre,3}$, $\mathsf{seed}^{\Delta}_3$ and $\{\mathsf{state}'_{j,n,3}\}_{j\in\mathcal{C}}$ to the verifier.
– Round 2. *Challenge* for the online phase.
  The verifier runs round 2 of $\pi^f_{on}(w', \{\mathsf{state}'_{j,i,2}\}_{j\in\mathcal{C},i\in[n]})$ to send a random set $\mathcal{P} = \{p_j\}_{j\in\mathcal{C}}$ with $p_j \in [n-1]$ to the prover.
– Round 3. *Respond* to the challenge for the online phase.
  The prover runs round 3 of $\pi^f_{on}(w', \{\mathsf{state}'_{j,i,2}\}_{j\in\mathcal{C},i\in[n]})$ to generate the corresponding response $\mathsf{resp}^f_{on,2}$. Send $\mathsf{resp}^f_{on,2}$ to the verifier.

**Verification Strategy**

1. For each unopened instance $j \in \mathcal{C}$, the verifier uses $\mathsf{resp}^f_{on,2}$, $\hat{w}'_j$, $\mathsf{seed}^\Delta_2$ and $\mathsf{state}'_{j,n,2}$ to simulate the MPC protocol for $C'(w') = y$ and recover the openings of $\mathsf{com}^f_{on,2}$ and $\mathsf{com}^f_{pre,2}$. Note that $\hat{w}'_j$, $\mathsf{seed}^\Delta_2$ and $\mathsf{state}'_{j,n,2}$ can be extracted from $\mathsf{vstate}$.
2. The verifier checks the output of the simulation of the MPC protocol.
3. The verifier checks the consistency of $\mathsf{com}^f_{pre,2}$, $\mathsf{com}^f_{on,2}$ and $\mathsf{com}_{on,2}$.

**State Update** The prover and the verifier update their states as follows.

– Prover's state update: $\mathsf{pstate} = \{\hat{w}'_j\}_{j\in\mathcal{C}} || \mathsf{seed}^\Delta_3 || \{\mathsf{state}'_{j,i,3}\}_{j\in\mathcal{C}, i\in[n]}$.
– Verifier's state update: $\mathsf{vstate} = \{\hat{w}'_j\}_{j\in\mathcal{C}} || \mathsf{seed}^\Delta_3 || \{\mathsf{state}'_{j,n,3}\}_{j\in\mathcal{C}} || \mathsf{com}^f_{pre,3}$.

## 5.3 Security

**Theorem 1.** *Assume that $\pi^F$, the underlying commitment scheme and pseudo-random generator are secure, and $F$ has an extractable decomposition as $f \circ g$. Then $\Pi_{Res}$ is a resumable honest verifier zero-knowledge proof of knowledge.*

The proof of zero-knowledge in Theorem 1 is similar to that of [51], while we should consider the zero-knowledge property of all the sessions as a whole. For the proof of resumable knowledge soundness, we show the consistency between $w'$ of $\mathsf{session}(t)$ and $w$, and use the method of [51,7] to construct a witness extractor $\mathcal{E}$ for each session. Formal proof of Theorem 1 is in Appendix A.

**Parallel Repetition.** The soundness error $\xi_t$ of $\Pi_{Res,2}$ for $\mathsf{session}(t)$ may be higher than $\xi_1$. We could reduce $\xi_t$ with parallel executions of $\Pi_{Res,2}$ as follows.

– In round 3 of the online phase $\pi_{on,1}$ (or round 1 of $\pi_{on,2}$) for $\mathsf{session}(t-1)$, the prover repeats round 1 of $\pi^f_{pre}$ for $\ell$ times. In other words, the prover rerandomizes intermediate masked values for $\ell$ times with $\ell$ random $\{\Delta_i\}_{i\in[\ell]}$.
– For $\mathsf{session}(t)$, the prover and the verifier run $\Pi_{Res,2}$ for $\ell$ times with the corresponding masked values generated in the previous session, where the verifier needs to send $\ell$ random challenges in round 2 of $\pi_{on,2}$.

Indeed, the above method can be interpreted as compacting $\ell$ executions of $\Pi_{Res,2}$ for $\ell$ sessions into one session, which will not break the security of our resumable HVZKPoK due to the honest verifier setting. In this way, we can reduce $\xi_t$ to $\frac{1}{(n-1)^{\tau\cdot\ell}}$. By choosing appropriate $\ell$, $M$, $n$ and $\tau$, we can gain a better soundness error for $\mathsf{session}(t)$. (Note that there is a trade-off between the soundness error and the proof size.) The proof of the following theorem are similar to that of Theorem 1 and hence omitted.

**Theorem 2.** *Assume that $\pi^F$, the underlying commitment scheme and pseudo-random generator are secure, and $F$ has an extractable decomposition as $f \circ g$. Then $\Pi_{Res}$ with parallel executions is a resumable honest verifier zero-knowledge proof of knowledge.*

### 5.4 3-Round Resumable HVZKPoK

Our 5-round $\Pi_{Res,1}$ can be transformed into a 3-round protocol using the similar method of [51]. Recall that the main idea of their transformation [51] is to have the prover simulate the online phase for every instance in $[M]$ and commit to the resulting transcripts in the first round. Our 3-round transformation is similar to that of [51] with the modification that the prover needs to prepare the random masks of the next session for every instance in $[M]$. Such modification has no effect on the security of the initial session, as those random masks could be considered as redundant information if there are no subsequent sessions.

To illustrate the construction of the 3-round resumable HVZKPoK, we provide a concrete construction, in which $F$ is instantiated with LowMC [2] as in [51]. While our concrete construction follows the general construction described in section 5, more optimizations including the ones proposed in [51,7] are taken into account due to the structure of LowMC. One of the advantages of using LowMC is that $f$ only consists of the KeyAddition operation in the last round. Thus we do not need to consider AND gates in circuit $C'$, which reduce the cost of the session resumption dramatically. More details of our 3-round resumable HVZKPoK $\Pi^3_{Res}$ are shown in Appendix B. The security proof of the following theorem is similar to that of Theorem 1 and hence omitted.

**Theorem 3.** *Assume that the underlying hash function, commitment scheme and pseudorandom generator are secure. Then $\Pi^3_{Res}$ is a resumable honest verifier zero-knowledge proof of knowledge.*

## 6 Resumable-Picnic

As in the previous works [23,51], our 3-round protocol can be transformed into a resumable *non-interactive* ZKPoK (NIZKPoK) using the Fiat-Shamir heuristic in each session, and the resulting NIZKPoK can be used to construct a stateful signature scheme. (We recall stateful signatures and the related security model in Appendix C.) More precisely, we instantiate $F$ with $\mathsf{Enc}(\cdot, 0^\kappa)$ for some symmetric encryption scheme $\mathsf{Enc}(\cdot,\cdot)$ in which the first input is the key and the second input is the plaintext. The signing key is a uniform $\mathsf{sk} \in \{0,1\}^\kappa$ and the verification key is $\mathsf{pk} = \mathsf{Enc}(\mathsf{sk}, 0^\kappa)$. By applying Fiat-Shamir heuristic to each session of our 3-round protocol for the relation $(\mathsf{pk}, \mathsf{sk}) \in R$, we can obtain a sequence of signatures. Specifically, the message being signed is included as input of the hash function, which is used to compute the challenge for each session in $\Pi^3_{Res}$. Denote the $t$-th signature as $\sigma_t$. Notice that the generation of $\sigma_t$ takes as inputs the message as well as the signer's "state" $\mathsf{pstate}_t$ which is generated from previous signature $\sigma_{t-1}$. We denote our stateful signature scheme as Resumable-Picnic. More details of Resumable-Picnic is present in Appendix C.1.

The resulting signature scheme is reminiscent of the chain-based stateful signature scheme [50], where the validity of $\sigma_t$ can be checked by verifying $\sigma_t$ as well as signatures $\{\sigma_i\}_{i<t}$ generated in all previous sessions. In the scenario where the verifier is fixed and can update the public state $\mathsf{vstate}_t$, the verifier

only needs to check the validity of $\sigma_t$ instead of all previous signatures. Different from the typical chain-based stateful signature, subsequent signatures in our stateful signature are much more efficient than the initial one.

**Theorem 4.** *Resumable-Picnic is strongly unforgeable under chosen message attacks in the QROM when* Com *is a collapse-binding commitment scheme and $H$ is a collapsing hash function.*

The proof of sketch of Theorem 4 is presented in Appendix D.

**Application**. One of the possible applications of Resumable-Picnic is in the blockchain setting, where each $\sigma_i$ can be stored in the corresponding block publicly. The verifier only needs to check the validity of $\sigma_t$ of the current block, since the validity of $\{\sigma_i\}_{i<t}$ in previous blocks are implied by the consistency of the underlying consensus protocol. More specifically, the signer can sign a transaction $\mathsf{tx}_1$ using Resumable-Picnic with pk and generate the signature $\sigma_1$. Then the miner generates a block $\mathsf{B}_i$ for a set of transactions with corresponding signatures, which includes $(\mathsf{tx}_1, \sigma_1)$. Afterwards, when the signer wants to sign another transaction $\mathsf{tx}_2$ with pk, he can generate $\sigma_2$ efficiently using the state $ss_1$. Due to the blockchain protocol, $(\mathsf{tx}_2, \sigma_2)$ will be included in some block, say $\mathsf{B}_j$, for $j > i$. If block $\mathsf{B}_i$ has been confirmed, which implies the validity of the signatures included in $\mathsf{B}_i$ have been confirmed by the majority of miners, the verifier of $\sigma_2$ does not need to check the validity of $\sigma_1$ any more. Due to the efficiency of session resumption, $\sigma_2$ is more efficient than the original Picnic[9]. (Note that there is usually a confirmation delay in most blockchain protocols. For instance, the confirmation delay of Bitcoin is about 6 blocks, which means a block is confirmed if it is followed by at least 6 blocks.)

## 7 Experimental Results and Comparison

We implement Resumable-Picnic using the same parameters as Picnic3 [49], and give an efficiency comparison with Picnic3. Our benchmarks run on a platform with an Intel Core i7-8700 CPU clocked at 3.2 GHz and 16GB RAM. The parameters are chosen as Picnic3 did which fits security level 1, 3, and 5 recommended by NIST. The comparison between Picnic3 and Resumable-Picnic are shown in Table 1. As shown in Table 1, although the cost of Resumable-Picnic's initial session is slightly higher than that of Picnic3, the efficiency of Resumable-Picnic for the subsequent sessions are improved dramatically. Compared with Picnic3 with security level 1, 3 and 5, the sign/verify time of Resumable-Picnic for session$(t > 2)$ is reduced to 9.2%/8.0%, 6.2%/5.6%, and 3.1%/3.3%, respectively, and the signature size is reduced to 38.1%, 37.2% and 36.0%, respectively.

[9] Recall that the generation and verification of the current signature only depends on the state of the last signature. Thus, we only need to keep the state of the latest signature.

**Table 1.** Comparison between Picnic3 and resumable-Picnic. "Size" denotes the signature size. The results are the median time for running 10000 times.

| Scheme | $M$ | $n$ | $\tau$ | $\ell$ | Sign (ms) | Verify (ms) | Size (Bytes) |
|---|---|---|---|---|---|---|---|
| Picnic3-Level 1 | 252 | 16 | 36 | | 71.68 | 51.37 | $12595 \pm 223$ |
| Resumable-Picnic [session(1)] | 252 | 16 | 36 | 1 | 99.99 | 76.23 | $14277 \pm 243$ |
| Resumable-Picnic [session(2)] | 252 | 16 | 36 | 1 | 8.31 | 4.78 | 4796 |
| Resumable-Picnic [session($t > 2$)] | 252 | 16 | 36 | 1 | 6.59 | 4.11 | 4796 |
| Picnic3-Level 3 | 419 | 16 | 52 | | 170.37 | 119.45 | $27104 \pm 455$ |
| Resumable-Picnic [session(1)] | 419 | 16 | 52 | 1 | 220.45 | 163.91 | $31166 \pm 466$ |
| Resumable-Picnic [session(2)] | 419 | 16 | 52 | 1 | 13.15 | 7.90 | 10088 |
| Resumable-Picnic [session($t > 2$)] | 419 | 16 | 52 | 1 | 10.60 | 6.67 | 10088 |
| Picnic3-Level 5 | 601 | 16 | 68 | | 487.45 | 290.22 | $48716 \pm 721$ |
| Resumable-Picnic [session(1)] | 601 | 16 | 68 | 1 | 512.56 | 332.24 | $55043 \pm 673$ |
| Resumable-Picnic [session(2)] | 601 | 16 | 68 | 1 | 18.28 | 11.26 | 17536 |
| Resumable-Picnic [session($t > 2$)] | 601 | 16 | 68 | 1 | 14.97 | 9.52 | 17536 |

## 8 Compressed 1-out-of-$N$ Proof and Ring Signatures

[26] provides a novel method of the one-out-of-$N$ proof for the relation $R_{\mathrm{OR}}$ defined by $(x_1, \ldots, x_N \in L_R; w) \in R_{\mathrm{OR}} \iff \exists t \in [N], s.t. (x_t, w) \in R$. By applying the parallel version of $\Pi_{Res,2}$ described in section 5.3 to the CDS method [26], we can get a compressed one-out-of-$N$ proof when the $N$ statements share the same circuit. The main idea is that, for the $N-1$ statements which the prover does not know the witness, the prover runs the simulator of the resumable HVZKPoK $\Pi_{Res,2}$ for the partial circuit $C'$ in parallel. Hence, most transcripts of the simulation for the $N-1$ statements can be removed. Furthermore, based on our compressed one-out-of-$N$ proof, we can construct a ring signature from symmetric-key primitives. More details of our compressed one-out-of-$N$ proof and the resulting ring signature are presented in Appendix F and Appendix G.

Using the same parameter set of Picnic2, we make a comparison between the ring signature of [51] and ours in Table 2. It shows that the size of our ring signature is smaller than that of [51] when the ring size is less than $2^6$. In particular, it is just about 1/3 of the ring signature size of [51] when the ring size is less than $2^4$.

**Table 2.** Comparison between ring signature [51] and our work.

| Ring size | 2 | $2^2$ | $2^3$ | $2^4$ | $2^5$ | $2^6$ | $2^7$ |
|---|---|---|---|---|---|---|---|
| $\lvert\sigma\rvert$ ([51]) | 70KB | 106KB | 142KB | 177KB | 213KB | 249KB | 285KB |
| $\lvert\sigma\rvert$ (Ours) | 21KB | 30KB | 47KB | 82KB | 151KB | 290KB | 567KB |

**Acknowledgements**. We would like to thank the anonymous reviewers for their insightful and helpful comments. Handong Zhang, Puwen Wei, Jinsong Li, Wei Wang and Guoxiao Liu were supported by the National Key Research and Development Program of China (Grant No. 2018YFA0704702), Shandong Provincial Key Research and Development Program (Major Scientific and Technological Innovation Project) (Grant No.2019JZZY010133) and Shandong Provincial Natural Science Foundation (Grant No. ZR2020MF053). Haiyang Xue was supported by the National Natural Science Foundation of China (Grant No. 62172412). Yi Deng was supported by the National Natural Science Foundation of China (Grant No. 61932019 and No. 61772522), the Key Research Program of Frontier Sciences, CAS (Grant No. QYZDB-SSW-SYS035) and Natural Science Foundation of Beijing (Grant No. M22003).

## References

1. Abe, M., Ambrona, M., Bogdanov, A., Ohkubo, M., Rosen, A.: Non-interactive composition of sigma-protocols via share-then-hash. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12493, pp. 749–773. Springer, Cham (2020), https://doi.org/10.1007/978-3-030-64840-4_25
2. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 430–454. Springer, Berlin, Heidelberg (2015), https://doi.org/10.1007/978-3-662-46800-5_17
3. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: ACM CCS 2017. p. 2087–2104. ACM Press, New York (2017), https://doi.org/10.1145/3133956.3134104
4. Aranha, D.F., Orlandi, C., Takahashi, A., Zaverucha, G.: Security of hedged Fiat–Shamir signatures under fault attacks. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 644–674. Springer, Cham (2020), https://doi.org/10.1007/978-3-030-45721-1_23
5. Attema, T., Cramer, R., Fehr, S.: Compressing proofs of k-out-of-n partial knowledge. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. pp. 65–91. Springer, Cham (2021), https://doi.org/10.1007/978-3-030-84259-8_3
6. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber. NIST PQC Round **3**, 4 (2020)
7. Baum, C., Nof, A.: Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. LNCS, vol. 12110, pp. 495–526. Springer, Cham (2020), https://doi.org/10.1007/978-3-030-45374-9_17
8. Baum, C., de Saint Guilhem, C.D., Kales, D., Orsini, E., Scholl, P., Zaverucha, G.: Banquet: Short and fast signatures from AES. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 11959, pp. 266–297. Springer, Cham (2021), https://doi.org/10.1007/978-3-030-75245-3_11
9. Bellare, M., Goldwasser, S.: New paradigms for digital signatures and message authentication based on non-interactive zero knowledge proofs. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 194–211. Springer, New York (1990), https://doi.org/10.1007/0-387-34805-0_19

10. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 701–732. Springer, Cham (2019), https://doi.org/10.1007/978-3-030-26954-8_23
11. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Snarks for c: Verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 90–108. Springer, Berlin, Heidelberg (2013), https://doi.org/10.1007/978-3-642-40084-1_6
12. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for r1cs. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019), https://doi.org/10.1007/978-3-030-17653-2_4
13. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: 23rd USENIX Security Symposium. pp. 781–796. USENIX Association, San Diego, CA (2014), https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson
14. Blum, M., De Santis, A., Micali, S., Persiano, G.: Noninteractive zero-knowledge. SIAM Journal on Computing **20**(6), 1084–1118 (1991), https://doi.org/10.1137/0220068
15. Boneh, D., Eskandarian, S., Fisch, B.: Post-quantum epid signatures from symmetric primitives. In: Matsui, M. (ed.) CT-RSA 2019. pp. 251–271. Springer, Cham (2019), https://doi.org/10.1007/978-3-030-12612-4_13
16. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 327–357. Springer, Berlin, Heidelberg (2016), https://doi.org/10.1007/978-3-662-49896-5_12
17. Bootle, J., Groth, J.: Efficient batch zero-knowledge arguments for low degree polynomials. In: Abdalla, M., Dahab, R. (eds.) PKC 2018. pp. 561–588. Springer, Cham (2018), https://doi.org/10.1007/978-3-319-76581-5_19
18. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: TCC 2020. LNCS, vol. 12551, pp. 1–18. Springer (2020), https://doi.org/10.1007/978-3-030-64378-2_1
19. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334 (2018). https://doi.org/10.1109/SP.2018.00020
20. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge (extended abstract). In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing. p. 235–244. STOC 2000, ACM, New York (2000), https://doi.org/10.1145/335305.335334
21. Chase, M., Derler, D., Goldfeder, S., Katz, J., Kolesnikov, V., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Wang, X., et al.: The picnic signature scheme, design document v2. 1 (2019)
22. Chase, M., Derler, D., Goldfeder, S., Katz, J., Kolesnikov, V., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Wang, X., Zaverucha, G.: The picnic signature scheme, design document v2. 2. Available at https://microsoft.github.io/Picnic/ (2020)
23. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: ACM CCS 2017. p. 1825–1842. ACM Press, New York (2017), https://doi.org/10.1145/3133956.3133997

24. Chiesa, A., Tromer, E.: Proof-carrying data and hearsay arguments from signature cards. In: Yao, A.C. (ed.) Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings. pp. 310–331. Tsinghua University Press (2010), http://conference.iiis.tsinghua.edu.cn/ICS2010/content/papers/25.html
25. Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile verifiable computation. In: 2015 IEEE Symposium on Security and Privacy. pp. 253–270 (2015). https://doi.org/10.1109/SP.2015.23
26. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. pp. 174–187. Springer, Berlin, Heidelberg (1994), https://doi.org/10.1007/3-540-48658-5_19
27. Deng, Y., Feng, D., Goyal, V., Lin, D., Sahai, A., Yung, M.: Resettable cryptography in constant rounds – the case of zero knowledge. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. pp. 390–406. Springer Berlin Heidelberg, Berlin, Heidelberg (2011), https://doi.org/10.1007/978-3-642-25385-0_21
28. Derler, D., Ramacher, S., Slamanig, D.: Post-quantum zero-knowledge proofs for accumulators with applications to ring signatures from symmetric-key primitives. In: Lange, T., Steinwandt, R. (eds.) PQCrypto 2018. pp. 419–440. Springer, Cham (2018), https://doi.org/10.1007/978-3-319-79063-3_20
29. Ding, J., Chen, M.S., Petzoldt, A., Schmidt, D., Yang, B.Y.: Rainbow. NIST PQC Round **3**, 4 (2020)
30. Dinur, I., Nadler, N.: Multi-target attacks on the picnic signature scheme and related protocols. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 699–727. Springer, Cham (2019), https://doi.org/10.1007/978-3-030-17659-4_24
31. Dodis, Y., Kiayias, A., Nicolosi, A., Shoup, V.: Anonymous identification in ad hoc groups. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. pp. 609–626. Springer, Berlin, Heidelberg (2004), https://doi.org/10.1007/978-3-540-24676-3_36
32. Don, J., Fehr, S., Majenz, C., Schaffner, C.: Security of the Fiat-Shamir transformation in the quantum random-oracle model. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11693, pp. 356–383. Springer, Cham (2019), https://doi.org/10.1007/978-3-030-26951-7_13
33. Don, J., Fehr, S., Majenz, C., Schaffner, C.: Security of the fiat-shamir transformation in the quantum random-oracle model. Cryptology ePrint Archive, Report 2019/190 (2019), https://eprint.iacr.org/2019/190
34. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: STOC 1998. p. 409–418. ACM Press, New York (1998), https://doi.org/10.1145/276698.276853
35. Feige, U., Lapidot, D., Shamir, A.: Multiple noninteractive zero knowledge proofs under general assumptions. SIAM Journal on Computing **29**(1), 1–28 (1999), https://doi.org/10.1137/S0097539792230010
36. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Berlin, Heidelberg (1987), https://doi.org/10.1007/3-540-47721-7_12
37. Fischlin, M., Harasser, P., Janson, C.: Signatures from sequential-OR proofs. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12107, pp. 212–244. Springer, Cham (2020), https://doi.org/10.1007/978-3-030-45727-3_8

38. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Berlin, Heidelberg (2013), https://doi.org/10.1007/978-3-642-38348-9_37
39. Giacomelli, I., Madsen, J., Orlandi, C.: Zkboo: Faster zero-knowledge for boolean circuits. In: 25th USENIX Security Symposium. pp. 1069–1083. USENIX Association, Austin, TX (2016), https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/giacomelli
40. Goel, A., Green, M., Hall-Andersen, M., Kaptchuk, G.: Stacking sigmas: A framework to compose $\sigma$-protocols for disjunctions. Cryptology ePrint Archive, Report 2021/422 (2021), https://ia.cr/2021/422
41. Goel, A., Green, M., Hall-Andersen, M., Kaptchuk, G.: Efficient set membership proofs using mpc-in-the-head. Proceedings on Privacy Enhancing Technologies **2022**(2), 304–324 (2022). https://doi.org/doi:10.2478/popets-2022-0047, https://doi.org/10.2478/popets-2022-0047
42. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In: SFCS 1986. pp. 174–187. IEEE Computer Society Press (1986). https://doi.org/10.1109/SFCS.1986.47
43. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1), 186–208 (1989), https://doi.org/10.1137/0218012
44. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Berlin, Heidelberg (2010), https://doi.org/10.1007/978-3-642-17373-8_19
45. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Berlin, Heidelberg (2016), https://doi.org/10.1007/978-3-662-49896-5_11
46. Groth, J., Ostrovsky, R., Sahai, A.: Perfect non-interactive zero knowledge for NP. In: Vaudenay, S. (ed.) EUROCRYPT 2006. pp. 339–358. Springer, Berlin, Heidelberg (2006), https://doi.org/10.1007/11761679_21
47. Henry, R., Goldberg, I.: Batch proofs of partial knowledge. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. pp. 502–517. Springer, Berlin, Heidelberg (2013), https://doi.org/10.1007/978-3-642-38980-1_32
48. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: STOC 2007. p. 21–30. ACM Press, New York (2007), https://doi.org/10.1145/1250790.1250794
49. Kales, D., Zaverucha, G.: Improving the performance of the picnic signature scheme. Cryptology ePrint Archive, Report 2020/427 (2020), https://eprint.iacr.org/2020/427
50. Katz, J.: Digital signatures. Springer Science & Business Media (2010)
51. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: ACM CCS 2018. p. 525–537. ACM Press, New York (2018), https://doi.org/10.1145/3243734.3243805
52. Kiltz, E., Lyubashevsky, V., Schaffner, C.: A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 552–586. Springer, Cham (2018), https://doi.org/10.1007/978-3-319-78372-7_18
53. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252 (2013). https://doi.org/10.1109/SP.2013.47

54. Peng, K., Bao, F.: Batch zk proof and verification of or logic. In: Yung, M., Liu, P., Lin, D. (eds.) Inscrypt 2008. Springer, Berlin, Heidelberg (2008), https://doi.org/10.1007/978-3-642-01440-6_13
55. Rescorla, E., Dierks, T.: The transport layer security (tls) protocol version 1.3. RFC 8446, DOI 10.17487/RFC8446, August 2018 (2018)
56. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Boyd, C. (ed.) ASIACRYPT 2001. pp. 552–565. Springer, Berlin, Heidelberg (2001), https://doi.org/10.1007/3-540-45682-1_32
57. de Saint Guilhem, C.D., De Meyer, L., Orsini, E., Smart, N.P.: BBQ: Using AES in picnic signatures. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 669–692. Springer, Cham (2020), https://doi.org/10.1007/978-3-030-38471-5_27
58. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) Theory of Cryptography. pp. 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), https://doi.org/10.1007/978-3-540-78524-8_1
59. Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zk-snarks without trusted setup. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 926–943 (2018). https://doi.org/10.1109/SP.2018.00060

## A Proof of Theorem 1

*Proof.* **Completeness.** This property follows from the correctness of the underlying MPC protocol $\Pi$ used in $\pi^F$ and $\pi^f$.
**Resumable Honest Verifier Zero-Knowledge.** We need to consider the simulator for all $q(\kappa)$ sessions instead of only one, where the simulation for the transcripts generated by $\pi^F$ and $\pi^f$ follows the idea of [51]. Let $\mathsf{Sim}_\Pi$ denotes the simulator of the MPC protocol $\Pi$. The simulator $\mathsf{Sim}$ of $\mathit{\Pi}_{Res}$ is described as follows.

- Simulation for initial session $\mathsf{session}(1)$.
  1. $\mathsf{Sim}$ chooses random $\mathcal{C}$ and $\mathcal{P}$ as the challenge for the preprocessing phase and the online phase respectively.
  2. For each instance $j \notin \mathcal{C}$, $\mathsf{Sim}$ prepares $\lambda_j$ using $\{\mathsf{state}_{j,i,1}\}_{i\in[n]}$ and generates the corresponding $\mathsf{resp}^F_{pre,1}$ as an honest prover would do in the preprocessing phase.
  3. For each instance $j \in \mathcal{C}$, $\mathsf{Sim}$ chooses a random masked input for the MPC protocol and $n-1$ random states for $n-1$ parties determined by $\mathcal{P}$. Then, $\mathsf{Sim}$ runs $\mathsf{Sim}_\Pi$ to simulate the views of the $n$ parties during the MPC protocol and computes corresponding $\mathsf{com}^F_{on,1}$. Notice that $\mathsf{Sim}$ can get the corresponding intermediate masked value $\hat{w}'_j$ for each instance $j \in \mathcal{C}$ from the simulated views. As mentioned in section 2.1, the indistinguishability between the simulated execution of $\mathsf{Sim}_\Pi$ and the real execution relies on the security of the underlying PRG.
  4. $\mathsf{Sim}$ computes $\mathsf{com}^F_{pre,1}$ and $\mathsf{resp}^F_{on,1}$ according to the transcripts generated in step 2 and 3. For the generation of $\mathsf{com}^F_{pre,1}$, the state of the party in $\mathcal{P}$ of each instance can be set by 0-string with appropriate length.
  5. $\mathsf{Sim}$ randomly chooses $\mathsf{seed}^\Delta_2$ and $\{\mathsf{state}'_{j,i,2}\}_{j\in\mathcal{C},i\in[n]}$, and computes the corresponding commitment $\mathsf{com}^f_{pre,2}$. Generate $\mathsf{com}_{on,1}$ as the commitment to $\mathsf{com}^F_{on,1}||\mathsf{com}^f_{pre,2}||\mathsf{seed}^\Delta_2||\{\mathsf{state}'_{j,n,2}\}_{j\in\mathcal{C}}$.
- Simulation for subsequent session $\mathsf{seesion}(t)$, where $1 < t \leq q(\kappa)$.
  1. $\mathsf{Sim}$ chooses a random $\mathcal{P}$ as the challenge for the online phase.
  2. For each instance $j \in \mathcal{C}$, $\mathsf{Sim}$ computes the rerandomized intermediate masked input $\hat{w}'_j \oplus \Delta_j$, in which $\hat{w}'_j$ is the intermediate masked value of $\mathsf{seesion}(1)$ and $\Delta_j$ is generated by $\mathsf{seed}^\Delta_t$. Note that $\mathsf{Sim}$ has $n-2$ parties' states determined by $\mathcal{P}$. Then, $\mathsf{Sim}$ runs $\mathsf{Sim}_\Pi$ to simulate the views of $n$ parties during the MPC protocol, and computes corresponding $\mathsf{com}^f_{on,t}$.
  3. $\mathsf{Sim}$ randomly chooses $\mathsf{seed}^\Delta_{t+1}$ and $\{\mathsf{state}'_{j,i,t+1}\}_{j\in\mathcal{C},i\in[n]}$, and computes the corresponding commitment $\mathsf{com}^f_{pre,t+1}$. Generate $\mathsf{com}_{on,t}$ as the commitment to $\mathsf{com}^f_{on,t}||\mathsf{com}^f_{pre,t+1}||\mathsf{seed}^\Delta_{t+1}||\{\mathsf{state}'_{j,n,t+1}\}_{j\in\mathcal{C}}$.

Following a standard hybrid argument, we have that the transcript generated by $\mathsf{Sim}$ is computationally indistinguishable from that of a real protocol, where the indistinguishability relies on the indistinguishability of the simulated transcripts generated by $\mathsf{Sim}_\Pi$ and the hiding property of the commitment scheme.

**Resumable Knowledge Soundness.** The proof of the resumable knowledge soundness is similar to that of [51,7], except that we need to show that there exists a witness extractor $\mathcal{E}$ for each session, especially the resumed session.

We first show the soundness error $\xi(M,n,\tau)$. Since $\Pi_{Res,1}$ is similar to that of the original KKW except additional processing for the masks of the next session. The soundness error $\xi_1$ of $\Pi_{Res,1}$ is the same as that of [51]. That is,

$$\xi_1(M,n,\tau) = \max_{0\le c\le\tau}\left\{\frac{\binom{M-c}{M-\tau}}{\binom{M}{M-\tau}\cdot n^{\tau-c}}\right\},$$

where $c$ denotes the number of preprocessing emulations where the malicious prover cheats.

On the soundness error of $\Pi_{Res,2}$, recall the soundness game mentioned in section 3, where the malicious prover can invoke the "honest" prover to interact with the verifier for polynomially-many sessions, say $\mathsf{session}(1),\ldots,\mathsf{session}(t-1)$ for $1<t\le q(\kappa)$, and tries to convince the verifier in $\mathsf{session}(t)$ without the help of the "honest" prover. Note that the masks for $\mathsf{session}(t)$ are generated by the honest prover in $\mathsf{session}(t-1)$. So a malicious prover of session $t$ can cheat only in the online phase, where he must cheat in one of the views of the $n-1$ parties. Thus, the probability that the prover will not be detected in $\Pi_{Res,2}$ is

$$\xi_t(M,n,\tau) = \frac{1}{(n-1)^\tau}.$$

Therefore, we have $\xi(M,n,\tau) = \max\{\xi_1(M,n,\tau),\xi_t(M,n,\tau)\}$, for any $1<t\le q(\kappa)$.

Next, we proceed to prove the resumable knowledge soundness property by showing how to construct $\mathcal{E}$ to extract a valid witness for each session. As explained above, the proof of knowledge soundness in [7] can be applied to $\Pi_{Res,1}$ directly. We focus on $\Pi_{Res,2}$ of $\mathsf{session}(t)$, where $1<t\le q(\kappa)$. For simplicity we assume that the commitment scheme is perfectly binding.

We first prove that if the success probability of cheating $\delta_t(x) > \xi_t(M,n,\tau)$, then there exists at least one MPC instance of $\mathcal{C}$, where the prover has committed to a valid intermediate value $w'$. Considering the deterministic prover with fixed random tape, let $\mathbf{v}$ be a 0/1-vector with length $(n-1)^\tau$, where each entry corresponds to a possible challenge for the online phase of $\mathcal{V}^{(t)}$ and 1 denotes the event of success. Hence, we have that $\delta_t(x)$ is the fraction of '1' entries in $\mathbf{v}$ and the number of '1' entries in $\mathbf{v}$ is higher than 1 due to $\delta_t(x) > \xi_t(M,n,\tau) = \frac{1}{(n-1)^\tau}$. That is, there must exist two accepting transcripts with different challenges $\{p_j\}_{j\in\mathcal{C}}$ and $\{p'_j\}_{j\in\mathcal{C}}$ such that $p_j \neq p'_j$ for an MPC instance $j$. That means all the views of the parties in instance $j$ are correct and the witness used in this instance must be a valid intermediate value $w'$.

However, since $f$ is just a part of $F$, it may be easy for a malicious prover to find a different $w^* \neq w'$ such that $f(w^*)=1$. It seems that any malicious prover who can find such a $w^*$ can cheat in the next session by computing $\lambda_{w^*} = w' \oplus \lambda_{w'} \oplus w^*$ and generating the corresponding $n$ shares of $\lambda_{w^*}\oplus\Delta$. (

$w' \oplus \lambda_{w'}$ can be extracted during the verification of the initial session.) Thanks to the binding property of the commitment $\mathsf{com}_{on,t}$ in $\pi_{cert}$, it is hard for the adversary to provide consistency proof using such $w^*$ and $\lambda_{w^*}$. For instance, in $\mathsf{session}(t-1)$, $\mathsf{com}_{on,t-1}$ is the commitment of $\mathsf{com}^f_{on,t-1}||\mathsf{com}^f_{pre,t}||\mathsf{seed}^{\Delta}_t$ $||\{\mathsf{state}'_{j,n,t}\}_{j\in\mathcal{C}}$, where ($\mathsf{com}_{on,t-1}$, $\mathsf{com}^f_{pre,t}$, $\mathsf{seed}^{\Delta}_t$, $\{\mathsf{state}'_{j,n,t}\}_{j\in\mathcal{C}}$) are public. The rerandomized mask for $\mathsf{session}(t)$, say $\lambda_{w'} \oplus \Delta$, is determined by ($\mathsf{com}^f_{pre,t}$, $\mathsf{seed}^{\Delta}_t$, $\{\mathsf{state}'_{j,n,t}\}_{j\in\mathcal{C}}$) and is hard to be modified due to $\mathsf{com}_{on,t-1}$. (The use of mask $\lambda_{w^*}$ such that $\lambda_{w^*} \neq \lambda_{w'} \oplus \Delta$ will be detected by checking the consistency of $\mathsf{com}_{on,t-1}$ and $\mathsf{com}^f_{pre,t}$.) Therefore, a malicious prover needs to (1) guess the challenge sent by the verifier successfully, which happens with probability $\frac{1}{n-1}$ for each instance, or (2) find $n-1$ random seeds which can be used to generate an $(n-1)$-out-of-$(n-1)$ secret-sharing of $\lambda_{w^*} \oplus \Delta \oplus [\lambda_{w'} \oplus \Delta]_n$, where each share is generated by running PRG with the corresponding random seed. This can be done with negligible probability assuming the underlying PRG is secure. Hence, $\mathsf{com}_{on,t-1}$ and $\mathsf{com}^f_{pre,t}$ guarantee the consistency of $w'$ in $\mathsf{session}(t)$ with $w$.

Next, we show how to extract the witness using two accepting transcripts with $\{p_j\}_{j\in\mathcal{C}}$ and $\{p'_j\}_{j\in\mathcal{C}}$ when the challenge for $j$ is different. Since $p_j \neq p'_j$, the transcripts with $p_j$ reveals $n-1$ shares of the masks of the intermediate masked input, whereas the transcripts with $p'_j$ reveals the remaining shares (Notice that the shares of the $n$-th party is public). Hence, we can get all the shares to recover the intermediate value $w'$. Due to the special property of the decomposition for $F$, the witness $w$ can be further extracted from $w'$.

To sum up, the extractor $\mathcal{E}$ is described as follows.

1. Run $\Pi_{Res,2}$ with the prover in session $t$ until the event of success happens, in order to find an '1' entry of the vector $\mathbf{v}$, where the corresponding challenge is $\{p_j\}_{j\in\mathcal{C}}$.
2. Run $\Pi_{Res,2}$ with the prover in session $t$ (using different challenges) until a different '1' entry is found, where the corresponding challenge is $\{p'_j\}_{j\in\mathcal{C}}$ such that $p_j \neq p'_j$.
3. Extract the witness $\omega$ in execution $j$ using the related transcripts with $\{p_j\}_{j\in\mathcal{C}}$ and $\{p'_j\}_{j\in\mathcal{C}}$. If $F(w) = y$, output $w$ and halt.

Let $\delta_t(x) = \xi_t(M, n, \tau) + \epsilon_t(x)$ for some $\epsilon_t(x) > 0$. The expected running time of the step 1 and 2 is $\frac{1}{\delta_t(x)} < \frac{1}{\epsilon_t(x)}$ and the running time of step 3 depends on the running time of $F(w)$ with common input $x$, which is supposed to be more efficient than step 1 and 2. Therefore, a valid witness can be extracted in $O(\frac{1}{\epsilon_t(x)})$ expected number of steps.

**Resumption efficiency**. $\Pi_{Res,2}$ consists of $\pi^f$ and the consistency proof $\pi_{cert}$. Since $\pi^f$ is a simplified KKW proof for the partial circuits of $F$ (without cut-and-choose), the complexity of $\pi^f$ is much smaller than that of the original KKW proof for $F$. Recall that $\pi_{cert}$ mainly consists of $\mathsf{com}_{on,2}$ and $\mathsf{seed}^{\Delta}_3$. So the complexity of $\pi_{cert}$ just takes a very small portion of $\pi^f$. Hence, although the overall complexity of $\Pi_{Res,2}$ depends on the concrete decomposition of $F$, $\Pi_{Res,2}$ is much efficient than that of the original KKW proof $\Pi'$ for $F$ in general. □

## B Concrete Construction of 3-Round Resumable HVZKPoK

The construction of 3-round resumable HVZKPoK $\Pi^3_{Res}$ is shown in Fig. 2, Fig. 3 and Fig. 4.

*Remark.* For the concrete construction of $\Pi_{Res,1}$ of $\mathsf{session}(1)$ as shown in Fig. 2, the prover chooses a random $\mathsf{salt}_1 \in \{0,1\}^\kappa$ to mitigate multi-target attack [30,7] and a random $\mathsf{seed}_2^\Delta \in \{0,1\}^\kappa$ to generate the rerandomized masks in step 1. For each instance $j \in [M]$, the main steps of the prover are explained as follows.

- In step 2(a), choose a random master seed $\mathsf{seed}^*_{j,1} \in \{0,1\}^\kappa$ to generate $\{\mathsf{seed}_{j,i,1}\}_{i\in[n]}$ and $\{\mathsf{seed}^{(u)}_{j,i,2}\}_{i\in[n],u\in[\ell]}$, which will be used to generate the states of the parties for the current session and the states of the parties of $\ell$ parallel executions for the next session, respectively.
- Commit to the states of the parties in step 2(d), where the resulting commitment is denoted by $\{\mathsf{com}_{j,i,1}\}_{i\in[n]}$, and compute the hash value $h_{j,1}$ of the corresponding commitments in step 2(f).
- Simulate the execution of the MPC protocol $\Pi$ among the parties in step 2(e), and commit to the resulting views of the parties using $h'_{j,1}$ in step 2(f).
- In step 2(g), use $\mathsf{seed}_2^\Delta$ to generate $\mathsf{seed}^\Delta_{j,2}$ which is used to rerandomize the masks for the $\ell$ parllel executions of the next session. Then, use $\{\mathsf{seed}^{(u)}_{j,i,2}\}_{i\in[n]}$ and $\mathsf{seed}^\Delta_{j,2}$ to generate the states of the parties of the parallel execution $u \in [\ell]$ for the next session, where the state of the $n$-th party for the parallel execution $u$ is denoted by $\{\mathsf{state}^{(u)}_{j,n,2}\}$.
- At the end of step 2, commit to $\{\mathsf{state}^{(u)}_{j,n,2}\}$ and $\mathsf{seed}_2^\Delta$ by computing $h_2^{sn}$, compute the commitment $h^*$ to $h_{j,1}$, $h'_{j,1}$ and $h_2^{sn}$ and send $h^*$ to the verifier.

For concrete construction of $\Pi_{Res,2}$ of $\mathsf{session}(t)$, where $1 < t \leq q(\kappa)$, the prover does not need to generate the states of the current session, and only needs to generate the states of the parties for $\ell$ parallel executions of the next session (step 3(a)(b)(e)), simulates the MPC protocol $\Pi$ among the parties by using the states generated from the previous session (step 3(c)), and then commits to the resulting states and views (step 3(d)(e)).

Furthermore, we remove $\mathsf{com}^f_{pre,2}$ and $\mathsf{com}^f_{pre,3}$ of $\pi_{cert}$ in the general construction in order to improve the efficiency. We show that such modification does not break the soundness due to the special structure of LowMC. Following the analysis in the proof of Theorem 1, $\lambda_{w'} \oplus \Delta$ cannot be determined "uniquely" if $\mathsf{com}^f_{pre,2}$ or $\mathsf{com}^f_{pre,3}$ is not provided. In this case, the adversary is able to compute $w^* = \hat{w}' \oplus \Delta \oplus \lambda_{w^*}$ such that $f(w^*) = y$, where $\lambda_{w^*}$ is generated "honestly". That is, the adversary chooses $n-1$ random seeds to generate the corresponding parties' shares $\{[\lambda_{w^*}]_i\}_{i\in[n-1]}$ using PRG and computes $\lambda_{w^*} = [\lambda_{w^*}]_1 \oplus \cdots \oplus [\lambda_{w^*}]_{n-1} \oplus [\lambda'_{w'}]_n$, which is indistinguishable from a random string assuming PRG is secure. So $w^*$ is indistinguishable from a random string. Notice that, for a random $w^*$ and fixed $y$, $f(w^*) = y$ holds only with negligible probability, since $f$ only consists of the KeyAddition operations. Therefore,

**Parameters:** Let $\kappa$ denote the security parameter. Let $H : \{0,1\}^* \rightarrow \{0,1\}^{2\kappa}$, $\mathsf{Com}$ and $\mathsf{PRG}$ denote hash function, commitment scheme and pseudorandom generator, respectively. $M, n, \tau$ and $\ell$ denote the number of instances, the number of the parties, the number of instances whose preprocessing phase will not be opened and the number of the parallel executions, respectively.
**Inputs:** Both the prover and the verifier have the statement $(C, y)$ with $C'$ as the partial circuit of $C$. The prover holds $w$ such that $C(w) = y$. The prover and the verifier initialize $t = 1$, $\mathsf{pstate}_t = \perp$ and $\mathsf{vstate}_t = \perp$, respectively.
**Round 1:**

1. The prover chooses random $\mathsf{salt}_t \in \{0,1\}^\kappa$ and $\mathsf{seed}^{\Delta}_{t+1} \in \{0,1\}^\kappa$.
2. If $t = 1$, for each $j \in [M]$, the prover does:
   (a) Choose a uniform $\mathsf{seed}^*_{j,t} \in \{0,1\}^\kappa$ as the seed of $\mathsf{PRG}$ to generate
   $(\mathsf{seed}_{j,1,t}, r_{j,1,t}), \ldots, (\mathsf{seed}_{j,n,t}, r_{j,n,t})$,
   $\mathsf{seed}^{(1)}_{j,1,t+1}, \ldots, \mathsf{seed}^{(1)}_{j,n,t+1}$,
   $\vdots$
   $\mathsf{seed}^{(\ell)}_{j,1,t+1}, \ldots, \mathsf{seed}^{(\ell)}_{j,n,t+1}$.
   (b) Compute $\mathsf{aux}_{j,n,t} \in \{0,1\}^{|C|}$ as described in the text. Set $\mathsf{state}_{j,n,t} = \mathsf{seed}_{j,n,t}||\mathsf{aux}_{j,n,t}$ and $\mathsf{state}_{j,i,t} = \mathsf{seed}_{j,i,t}$ for each $i \in [n-1]$.
   (c) Generate $\mathsf{seed}^{\Delta}_{j,t+1}$ using $\mathsf{seed}^{\Delta}_{t+1}$.
   (d) For each $i \in [n]$, compute $\mathsf{com}_{j,i,t} = \mathsf{Com}(\mathsf{state}_{j,i,t}, r_{j,i,t}, \mathsf{salt}_t)$.
   (e) Simulate the online phase of the MPC protocol $\Pi$ for $C$ as follows:
      i. For each input wire $\alpha$ of $C$, using $\{\mathsf{seed}_{j,i,t}\}_{i\in[n]}$ to generate mask values $\{\lambda_{j,\alpha,t}\}$.
      ii. Compute the masked input $\{\hat{z}_{j,\alpha,t}\}$ as $\hat{z}_{j,\alpha,t} = w_\alpha \oplus \lambda_{j,\alpha,t}$, where $w_\alpha$ denotes the value of the input wire $\alpha$.
      iii. Evaluate $C$ by proceeding through the gates in topological order. For each party $P_i$, compute the broadcast message $\mathsf{msgs}_{j,i,t}$.
      iv. Denote the intermediate masked output of evaluating $C$ as $\{\hat{z}_{j,\alpha,inter}\}$ which is the input value of $C'$. Note that $\hat{z}_{j,\alpha,inter} = z_{j,\alpha,inter} \oplus \lambda_{j,\alpha,inter}$, where $\lambda_{j,\alpha,inter}$ is the mask value for each input wire $\alpha$ of $C'$.
   (f) Compute $h_{j,t} = H(\mathsf{com}_{j,1,t}, \ldots, \mathsf{com}_{j,n,t})$ and $h'_{j,t} = H(\{\hat{z}_{j,\alpha,t}\}, \mathsf{msgs}_{j,1,t}, \ldots, \mathsf{msgs}_{j,n,t}, \mathsf{salt}_t)$.
   (g) Use $\mathsf{seed}^{\Delta}_{j,t+1}$ to generate $\{\Delta^{(1)}_{j,\alpha,t+1}\}, \ldots, \{\Delta^{(\ell)}_{j,\alpha,t+1}\}$ for each input wire $\alpha$ of $C'$.
      i. For each $u \in [\ell]$, compute $\mathsf{aux}^{(u)}_{j,n,t+1} \in \{0,1\}^{|C'|+|C'_{in}|}$ as described in the text.
      ii. For each $u \in [\ell]$, set $\mathsf{state}^{(u)}_{j,i,t+1} = \mathsf{seed}^{(u)}_{j,i,t+1}$ for each $i \in [n-1]$; set $\mathsf{state}^{(u)}_{j,n,t+1} = \mathsf{seed}^{(u)}_{j,n,t+1}||\mathsf{aux}^{(u)}_{j,n,t+1}$. Compute $h^n_{j,t+1} = H(\mathsf{state}^{(1)}_{j,n,t+1}, \ldots, \mathsf{state}^{(\ell)}_{j,n,t+1}, \mathsf{salt}_t)$.

   Compute $h^{sn}_{t+1} = H(\mathsf{seed}^{\Delta}_{t+1}, h^n_{1,t+1}, \ldots, h^n_{M,t+1})$, $h_t = H(h_{1,t}, \ldots, h_{M,t})$ and $h'_t = H(h'_{1,t}, \ldots, h'_{M,t})$. Then, send $h^*_t = H(h_t, h'_t, h^{sn}_{t+1})$ to the verifier.

**Fig. 2.** Our 3-round resumable HVZK proof (Part 1)

3. If $t > 1$, for each $j \in \mathcal{C}$, the prover does:
   (a) Choose a uniform $\mathsf{seed}^*_{j,t} \in \{0,1\}^\kappa$ as the seed of $\mathsf{PRG}$ to generate
   $\mathsf{seed}^{(1)}_{j,1,t+1}, \ldots, \mathsf{seed}^{(1)}_{j,n,t+1}$,
   $\vdots$
   $\mathsf{seed}^{(\ell)}_{j,1,t+1}, \ldots, \mathsf{seed}^{(\ell)}_{j,n,t+1}$.
   (b) Generate $\mathsf{seed}^{\Delta}_{j,t+1}$ using $\mathsf{seed}^{\Delta}_{t+1}$.
   (c) Simulate the online phase of the MPC protocol $\Pi$ for $C'$ using $\{\mathsf{state}^{(u)}_{j,i,t}\}_{i\in[n]}$ for each $u \in [\ell]$ as follows:
      i. Compute the masked input $\{\hat{z}^{(u)}_{j,\alpha,t}\}$ of $C'$ as $\hat{z}^{(u)}_{j,\alpha,t} = \hat{z}_{j,\alpha,inter} \oplus \Delta^{(u)}_{j,\alpha,t}$.
      ii. Evaluate $C'$ by proceeding through the gates in topological order. For each party $P_i$, compute the broadcast message $\mathsf{msgs}^{(u)}_{j,i,t}$.
   (d) Compute $h'_{j,t} = H(\{\hat{z}_{j,\alpha,inter}\}, \mathsf{seed}^{\Delta}_{j,t}, \mathsf{msgs}^{(1)}_{j,1,t}, \ldots, \mathsf{msgs}^{(1)}_{j,n,t}, \ldots, \mathsf{msgs}^{(\ell)}_{j,1,t}, \ldots, \mathsf{msgs}^{(\ell)}_{j,n,t}, \mathsf{salt}_t)$.
   (e) Use $\mathsf{seed}^{\Delta}_{j,t+1}$ to generate $\{\Delta^{(1)}_{j,\alpha,t+1}\}, \ldots, \{\Delta^{(\ell)}_{j,\alpha,t+1}\}$ for each input wire $\alpha$ of $C'$.
      i. For each $u \in [\ell]$, compute $\mathsf{aux}^{(u)}_{j,n,t+1} \in \{0,1\}^{|C'|+|C'_{in}|}$ as described in the text.
      ii. For each $u \in [\ell]$, set $\mathsf{state}^{(u)}_{j,i,t+1} = \mathsf{seed}^{(u)}_{j,i,t+1}$ for each $i \in [n-1]$; set $\mathsf{state}^{(u)}_{j,n,t+1} = \mathsf{seed}^{(u)}_{j,n,t+1} || \mathsf{aux}^{(u)}_{j,n,t+1}$. Compute $h^n_{j,t+1} = H(\mathsf{state}^{(1)}_{j,n,t+1}, \ldots, \mathsf{state}^{(\ell)}_{j,n,t+1}, \mathsf{salt}_t)$.

   Let $\mathcal{C} = \{j_1, \ldots, j_\tau\}$. Compute $h^{sn}_{t+1} = H(\mathsf{seed}^{\Delta}_{t+1}, h^n_{j_1,t+1}, \ldots, h^n_{j_\tau,t+1})$ and $h'_t = H(h'_{j_1,t}, \ldots, h'_{j_\tau,t})$. Then, send $h^*_t = H(h'_t, h^{sn}_{t+1})$ to the verifier.

**Round 2:**

1. If $t = 1$, the verifier chooses a uniform $\tau$-sized set $\mathcal{C} \subset [M]$ and $\mathcal{P}_t = \{p_{j,t}\}_{j\in\mathcal{C}}$, where $p_{j,t} \in [n]$. Send $(\mathcal{C}, \mathcal{P}_t)$ to the prover.
2. If $t > 1$, the verifier chooses a uniform $\mathcal{P}_t = \{p^{(u)}_{j,t}\}_{j\in\mathcal{C},u\in[\ell]}$, where $p^{(u)}_{j,t} \in [n-1]$. Send $\mathcal{P}_t$ to the prover.

**Round 3:** The prover sends $\mathsf{salt}_t$ and $\mathsf{seed}^{\Delta}_{t+1}$ as well as the following values to the verifier.

1. If $t = 1$, for each $j \in [M]\backslash\mathcal{C}$, the prover sends $\mathsf{seed}^*_{j,t}, h'_{j,t}$ to the verifier. For each $j \in \mathcal{C}$, the prover sends $\{\mathsf{state}_{j,i,t}, r_{j,i,t}\}_{i\neq p_{j,t}}, \mathsf{com}_{j,p_{j,t},t}$, $\{\hat{z}_{j,\alpha,t}\}$, $\mathsf{msgs}_{j,p_{j,t},t}$ and $\{\mathsf{state}^{(u)}_{j,n,t+1}\}_{u\in[\ell]}$ to the verifier.
2. If $t > 1$, for each $j \in \mathcal{C}, u \in [\ell]$, the prover sends $\{\mathsf{state}^{(u)}_{j,i,t}\}_{i\in[n-1], i\neq p^{(u)}_{j,t}}$, $\mathsf{msgs}^{(u)}_{j,p^{(u)}_{j,t},t}$ and $\{\mathsf{state}^{(u)}_{j,n,t+1}\}$ to the verifier.

**Fig. 3.** Our 3-round resumable HVZK proof (Part 2)

**Verification:** The verifier accepts iff all the following checks succeed.

1. If $t = 1$:
   (a) For each $j \in \mathcal{C}$, use $\{\mathsf{state}_{j,i,t}, r_{j,i,t}\}_{i \neq p_{j,t}}$ and $\mathsf{salt}_t$ to compute $\{\mathsf{com}_{j,i,t}\}_{i \neq p_{j,t}}$. Then, compute $h_{j,t} = H(\mathsf{com}_{j,1,t}, \ldots, \mathsf{com}_{j,n,t})$.
   (b) For each $j \in \mathcal{C}$, use $\{\mathsf{state}^{(u)}_{j,n,t+1}\}_{u \in [\ell]}$ and $\mathsf{salt}_t$ to compute $h^n_{j,t+1} = H(\mathsf{state}^{(1)}_{j,n,t+1}, \ldots, \mathsf{state}^{(\ell)}_{j,n,t+1}, \mathsf{salt}_t)$.
   (c) For each $j \in [M] \backslash \mathcal{C}$, use $\mathsf{seed}^*_{j,t}$, $\mathsf{salt}_t$ and $\mathsf{seed}^{\Delta}_{t+1}$ to compute $h_{j,t}$ and $h^n_{j,t+1}$ as an honest prover would. Then compute $h_t = H(h_{1,t}, \ldots, h_{M,t})$ and $h^{sn}_{t+1} = H(\mathsf{seed}^{\Delta}_{t+1}, h^n_{1,t+1}, \ldots, h^n_{M,t+1})$.
   (d) For each $j \in \mathcal{C}$, simulate an execution of $\Pi$ for $C$ among $\{P_i\}_{i \neq p_{j,t}}$ using $\{\mathsf{state}_{j,i,t}\}_{i \neq p_{j,t}}$, masked inputs $\{\hat{z}_{j,\alpha,t}\}$ and $\mathsf{msgs}_{j,p_{j,t},t}$. This will yield $\{\mathsf{msgs}_{j,i,t}\}_{i \neq p_{j,t}}$, $\{\hat{z}_{j,\alpha,inter}\}$ and an output $y'$. Check that $y \overset{?}{=} y'$ and compute $h'_{j,t} = H(\{\hat{z}_{j,\alpha,t}\}, \mathsf{msgs}_{j,1,t}, \ldots, \mathsf{msgs}_{j,n,t}, \mathsf{salt}_t)$ as well as $h'_t = H(h'_{1,t}, \ldots, h'_{M,t})$.
   (e) Check that $H(h_t, h'_t, h^{sn}_{t+1}) \overset{?}{=} h^*_t$.
2. If $t > 1$:
   (a) For each $j \in \mathcal{C}$, use $\{\mathsf{state}^{(u)}_{j,n,t+1}\}_{u \in [\ell]}$ and $\mathsf{salt}_t$ to compute $h^n_{j,t+1} = H(\mathsf{state}^{(1)}_{j,n,t+1}, \ldots, \mathsf{state}^{(\ell)}_{j,n,t+1}, \mathsf{salt}_t)$. Then, compute $h^{sn}_{t+1} = H(\mathsf{seed}^{\Delta}_{t+1}, h^n_{j_1,t+1}, \ldots, h^n_{j_\tau,t+1})$.
   (b) For each $j \in \mathcal{C}, u \in [\ell]$, simulate an execution of $\Pi$ for $C'$ among $\{P_i\}_{i \neq p^{(u)}_{j,t}}$ using $\{\mathsf{state}^{(u)}_{j,i,t}\}_{i \neq p^{(u)}_{j,t}}$, masked inputs $\{\hat{z}^{(u)}_{j,\alpha,t}\}$ and $\mathsf{msgs}^{(u)}_{j,p^{(u)}_{j,t},t}$. This will yield $\{\mathsf{msgs}^{(u)}_{j,i,t}\}_{i \neq p^{(u)}_{j,t}}$ and an output $y'$. Check that $y \overset{?}{=} y'$ and compute $h'_{j,t} = H(\{\hat{z}_{j,\alpha,inter}\}, \mathsf{seed}^{\Delta}_{j,t}, \mathsf{msgs}^{(1)}_{j,1,t}, \ldots, \mathsf{msgs}^{(1)}_{j,n,t}, \ldots, \mathsf{msgs}^{(\ell)}_{j,1,t}, \ldots, \mathsf{msgs}^{(\ell)}_{j,n,t}, \mathsf{salt}_t)$ as well as $h'_t = H(h'_{j_1,t}, \ldots, h'_{j_\tau,t})$.
   (c) Check that $H(h'_t, h^{sn}_{t+1}) \overset{?}{=} h^*_t$.

**State Update:** After the $t$-th session, the prover and the verifier update their state for the subsequent session as follows.

1. The prover sets $\mathsf{pstate}_t = \{\mathsf{state}^{(u)}_{j,i,t+1}\}_{j \in \mathcal{C}, i \in [n], u \in [\ell]} || \{\mathsf{seed}^{\Delta}_{j,t+1}\}_{j \in \mathcal{C}} || \{\hat{z}_{j,\alpha,inter}\}_{j \in \mathcal{C}}$.
2. The verifier sets $\mathsf{vstate}_t = \{\mathsf{state}^{(u)}_{j,n,t+1}\}_{j \in \mathcal{C}, u \in [\ell]} || \mathsf{seed}^{\Delta}_{t+1} || \{\hat{z}_{j,\alpha,inter}\}_{j \in \mathcal{C}}$.
3. Both the prover and the verifier set $t = t + 1$.

**Fig. 4.** Our 3-round resumable HVZK proof (Part 3)

although $\mathsf{com}^f_{pre,2}$ and $\mathsf{com}^f_{pre,3}$ are removed, the knowledge soundness can still be guaranteed when instantiated with LowMC.

## C Stateful Signature Scheme

**Definition 3.** *[50] A stateful signature scheme is a tuple of probabilistic polynomial time (PPT) algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *which satisfies that:*

- *The key generation algorithm* $\mathsf{Gen}$ *takes as input a security parameter* $1^\kappa$ *and outputs* $(pk, sk, ss_0)$*, which are the public key, the private key and the signer's initial state, respectively.*
- *The signing algorithm* $\mathsf{Sign}$ *takes as input a private key* $sk$*, a message* $m \in \{0,1\}^*$*, and a state* $ss_{i-1}$*. It outputs a signature* $\sigma$ *and an updated state* $ss_i$*.*
- *The deterministic verification algorithm* $\mathsf{Vrfy}$ *takes as input a public key* $pk$*, a message* $m$*, a signature* $\sigma$ *and a state* $vs_{i-1}$*. It outputs a bit* $b$*, and an updated state* $vs_i$ *if* $b = 1$*. Here, suppose the verifier's initial state is* $vs_0$*.*

*We require that, for every* $(pk, sk, ss_0)$ *output by* $\mathsf{Gen}(1^\kappa)$ *and any message* $m_1, \ldots, m_q \in \{0,1\}^*$*, it holds that* $(1, vs_i) \leftarrow \mathsf{Vrfy}_{pk}(m_i, \sigma_i, vs_{i-1})$ *holds for every* $i \in \{1, \ldots, q\}$*, where* $(\sigma_i, ss_i) \leftarrow \mathsf{Sign}_{sk}(m_i, ss_{i-1})$*.*

In this paper, we consider the stateful signature where the verifier can hold a state in order to verify the signature efficiently. The initial state of the verifier is empty, i.e., $vs_0 = \perp$, and the state of the verifier $vs_i$ is updated by the verification of $(m_i, \sigma_i)$. Thus, the verification algorithm can be rewritten as $1/0 \leftarrow \mathsf{Vrfy}_{pk}(m_i, \sigma_i, \{(m_j, \sigma_j)\}_{j=1}^{i-1})$. The security of stateful signatures is defined by the experiment $\mathsf{SSig\text{-}Forge}_{\mathcal{A},\Pi}(\kappa)$ shown in Fig 5.

**Fig. 5.** $\mathsf{SSig\text{-}Forge}_{\mathcal{A},\Pi}(\kappa)$ Experiment

1. *Run* $\mathsf{Gen}$ *to generate* $(pk, sk, ss_0)$*.*
2. *Adversary* $\mathcal{A}$ *is given* $pk$*, and access to a stateful signing oracle* $\mathsf{SSign}_{sk}(\cdot)$*.* $\mathcal{A}$ *sets* $vs = vs_0$*. After each query,* $\mathcal{A}$ *and* $\mathsf{SSign}_{sk}(\cdot)$ *update the state* $vs$ *and* $ss$*, respectively.*
3. *Let* $\mathcal{Q} = \{m_i\}_{i=1}^q$ *denote the sequence of queries made by* $\mathcal{A}$*, in which* $q$ *denotes the number of queries made by* $\mathcal{A}$ *and* $q \leq q(\kappa)$*.* $\mathcal{A}$ *outputs* $\{(m'_i, \sigma'_i)\}_{i=1}^{q'}$*. Let* $\mathcal{Q}' = \{m'_i\}_{i=1}^{q'}$ denote the sequence of the messages $m'_1, \ldots, m'_{q'}$*.*
4. *The experiment outputs* 1 *if (1)* $\mathsf{Vrfy}_{pk}(m'_{q'}, \sigma'_{q'}, \{(m'_i, \sigma'_i)\}_{i=1}^{q'-1}) = 1$ *and (2)* $\mathcal{Q}' \neq \mathcal{Q}$ *and* $\mathcal{Q}'$ *is not a predecessor of* $\mathcal{Q}$*.*

**Definition 4.** *A stateful signature scheme* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is* existentially unforgeable under an adaptive chosen-messgga attack*, or* EUF-CMA*, if for all PPT adversary* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}$ *such that:*

$$\Pr[\mathsf{SSig\text{-}Forge}_{\mathcal{A},\Pi}(\kappa) = 1] \leq \mathsf{negl}(\kappa).$$

If we set $\mathcal{Q} = \{(m_i, \sigma_i)\}_{i=1}^q$ and $\mathcal{Q}' = \{(m'_i, \sigma'_i)\}_{i=1}^{q'}$, the corresponding experiment is denoted as $\mathsf{SSig\text{-}Forge}'_{\mathcal{A},\Pi}(\kappa)$. Then we have the definition of strongly EUF-CMA.

**Definition 5.** *A stateful signature scheme* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is* strongly unforgeable under an adaptive chosen-messga attack, *or* sEUF-CMA, *if for all PPT adversary* $\mathcal{A}$, *there is a negligible function* negl *such that:*

$$\Pr[\mathsf{SSig\text{-}Forge}'_{\mathcal{A},\Pi}(\kappa) = 1] \leq \mathsf{negl}(\kappa).$$

### C.1 Construction of Resumable-Picnic

Resumable-Picnic consists of three probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$, which are defined as follows:

- $\mathsf{Gen}(\kappa)$: The key generation algorithm takes as input the security parameter $\kappa$. Choose a uniform signing key $\mathsf{sk} \in \{0,1\}^\kappa$ and generate the verification key $\mathsf{pk} = \mathsf{Enc}(\mathsf{sk}, 0^\kappa)$, where $\mathsf{Enc}$ denotes a symmetric encryption scheme, e.g., LowMC.
- $\mathsf{Sign}_{\mathsf{sk}}(m_i, ss_{i-1})$: The signing algorithm takes as inputs the signing key $\mathsf{sk}$, the message $m_i \in \{0,1\}^*$ and the signer's state $ss_{i-1}$ (where the initial state $ss_0 = \perp$). Set $\mathsf{pstate}_{i-1} = ss_{i-1}$. Run the prover's strategy of the resumable NIZKPoK for $\mathsf{pk} = \mathsf{Enc}(\mathsf{sk}, 0^\kappa)$ of $\mathsf{session}(i)$ to generate the proof, where $m_i$ is included as input of the hash function to compute the challenge. The resulting transcript of proof is the signature. Finally, it outputs a signature $\sigma_i$ and a signer's state $ss_i = \mathsf{pstate}_i$.
- $\mathsf{Vrfy}_{\mathsf{pk}}(m_i, \sigma_i, vs_{i-1})$: The verification algorithm takes as inputs the verification key $\mathsf{pk}$, the message $m_i$, the signature $\sigma_i$ and the verifier's state $vs_{i-1}$. Set $\mathsf{vstate}_{i-1} = vs_{i-1}$. Run the verifier's strategy of the resumable NIZKPoK of $\mathsf{session}(i)$ to verify the signature. Finally, it outputs $(1, vs_i)$ for acceptance, where the verifier's state $vs_i = \mathsf{vstate}_i$, or just 0 for rejection.

Notice that verifier's initial state $vs_0 = \perp$ and is updated by the verification algorithm. Thus, the verification algorithm can be rewritten as $1/0 \leftarrow \mathsf{Vrfy}_{\mathsf{pk}}(m_i, \sigma_i, \{(m_j, \sigma_j)\}_{j=1}^{i-1})$.

## D Proof of Sketch for Theorem 4

Let $\{m_i\}_{i=1}^q$ denote the messages queried by the adversary in the $\mathsf{SSig\text{-}Forge}'_{\mathcal{A}}$ experiment. The corresponding set of message/signature pairs are denoted as $\mathcal{Q} = \{(m_i, \sigma_i)\}_{i=1}^q$. Suppose $\mathcal{Q}' = \{(m'_i, \sigma'_i)\}_{i=1}^{q'}$ is the forgery output by the adversary $\mathcal{A}$. Consider the following two cases.

1. For each $i \in [q']$, $(m'_i, \sigma'_i) \in \mathcal{Q}$.
2. There exists some $i \in [q']$ such that $(m'_i, \sigma'_i) \notin \mathcal{Q}$.

Case 1: Since the structure of $\sigma_1$ is different from that of its subsequent signatures, $\mathcal{A}$ cannot make a forgery by changing the order of $(m_1, \sigma_1)$ or replacing $(m_1, \sigma_1)$ with $(m_i, \sigma_i)$ for some $i > 1$. Without loss of generality, we assume that $i_1$ is the least index such that $(m'_{i_1}, \sigma'_{i_1}) \neq (m_{i_1}, \sigma_{i_1})$ and $(m'_{i_1}, \sigma'_{i_1}) = (m_{i_2}, \sigma_{i_2})$ for some $i_2 \neq i_1$, where $1 < i_1 \leq q'$ and $1 < i_2 \leq q$. Here, the least index implies $(m'_i, \sigma'_i) = (m_i, \sigma_i)$ for $i < i_1$. Due to $\sigma'_{i_1} = \sigma_{i_2}$, we have $h^*_{i_1} = h^*_{i_2}$, where $h^*_{i_1}$ and $h^*_{i_2}$ are parts of $\sigma'_{i_1}$ and $\sigma_{i_2}$, respectively. Recall that $h^*_t = H(h'_t, h^{sn}_{t+1})$, for $t > 1$, and $h'_t$ is the hash value of the challenged parties' broadcast messages during the online phase and the random seed, say $\mathsf{seed}^{\Delta}_{j,t}$. Note that $\mathsf{seed}^{\Delta}_{j,t}$ is generated using $\mathsf{seed}^{\Delta}_{t}$. In $\mathsf{SSig\text{-}Forge}'_{\mathcal{A}}$, we have $\mathsf{seed}^{\Delta}_{i_1} \neq \mathsf{seed}^{\Delta}_{i_2}$ except with negligible probability, since $\mathsf{seed}^{\Delta}_{i_1}$ and $\mathsf{seed}^{\Delta}_{i_2}$ are chosen randomly by the signing oracle in $\mathsf{session}(i_1 - 1)$ and $\mathsf{session}(i_2 - 1)$, respectively. So we have $\mathsf{seed}^{\Delta}_{j,i_1} \neq \mathsf{seed}^{\Delta}_{j,i_2}$ except with negligible probability assuming the underlying PRG is secure. Therefore, $h^*_{i_1} = h^*_{i_2}$ implies a collision of the underlying hash functions, which contradicts the collapsing property.

Case 2: Let $\mathsf{Sig}[\mathit{\Pi}_{Res,1}]$ denote the signature scheme, which is constructed by applying Fiat-Shamir heuristic to $\mathit{\Pi}_{Res,1}$. $\mathsf{Sig}[\mathit{\Pi}_{Res,1}]$ is essentially the same as $\mathsf{Picnic}$, which is sEUF-CMA in the QROM. (The additional preprocessing data for the next session does not break the security of $\mathsf{Sig}[\mathit{\Pi}_{Res,1}]$ if the underlying hash function is secure.) Furthermore, if the state $\mathsf{vstate}_{i-1}$ is the public parameter (or generated by the trusted party), the signature for each session, say $\sigma_i$, can be considered as an one-time signature. Let $\mathsf{Sig}[\mathit{\Pi}_{Res,2}]$ denote the signature scheme constructed by applying Fiat-Shamir heuristic to $\mathit{\Pi}_{Res,2}$.

In fact, the forgery in case 2 implies a forgery for $\mathsf{Sig}[\mathit{\Pi}_{Res,1}]$ or $\mathsf{Sig}[\mathit{\Pi}_{Res,2}]$, which contradicts the security of $\mathsf{Sig}[\mathit{\Pi}_{Res,1}]$ or $\mathsf{Sig}[\mathit{\Pi}_{Res,2}]$. It remains to show that $\mathsf{Sig}[\mathit{\Pi}_{Res,2}]$ is one-time sEUF-CMA in the QROM, which can be proven using the same method of [32]. To apply [32, Corollary 26], the $\mathit{\Sigma}$-protocol $\mathit{\Pi}_{Res,2}$ should satisfy the following properties:

1. Be non-abort honest verifier zero-knowledge (naHVZK) [52]. This property is implied by Theorem 3, in which the commitments are randomized.
2. Have min-entropy $\alpha$ [52, Definition 2.6] which is polynomial in the security parameter. This property can be proven using [4, Lemma 17], which shows that $\mathit{\Pi}_{Res,2}$ has min-entropy $\alpha \geq 2\kappa + 256$.
3. Has quantum computational unique responses (CUR). Following the idea of [22, Lemma 6.7], we can show that $\mathit{\Pi}_{Res,2}$ has classical CUR. Then, the quantum CUR of $\mathit{\Pi}_{Res,2}$ can be proven under the assumptions that the hash functions are collapsing and the commitment scheme is collapse-binding. More details of the computational unique responses are shown in Appendix E.

As shown in Theorem 1, $\mathit{\Pi}_{Res,2}$ is a 2-sound PoK protocol. Hence, we can show that $\mathit{\Pi}_{Res,2}$ is computational proof of knowledge (PoK) in the QROM by applying [33, Theorem 25], which proves that a $\mathit{\Sigma}$-protocol with $t$-soundness (for some constant $t$) and quantum CUR is a computational proof of knowledge (PoK) in the QROM.

Finally, the security of $\mathsf{Sig}[\Pi_{Res,2}]$ can be proven by applying [33, Theorem 22], which says that a $\Sigma$-protocol with the PoK property, satisfying non-abort HVZK with $\alpha$ bits of min-entropy and CUR, is strongly unforgeable under chosen-message attacks.

## E Computational Unique Responses

**Definition 6 (Computational Unique Responses [52]).** *For a $\Sigma$-protocol $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{Ch}, \mathsf{V})$ with transcripts $(a, e, z)$, for all PPT adversaries $\mathcal{A}$, if it satisfies that,*

$$\Pr\left[\begin{matrix}\mathsf{V}(pk, a, e, z) = \mathsf{V}(pk, a, e, z') = 1 \\ \wedge z \neq z'\end{matrix} \,\middle|\, \begin{matrix}(pk, sk) \leftarrow \mathsf{Gen}(\kappa); \\ (a, e, z, z') \leftarrow \mathcal{A}(\kappa)\end{matrix}\right] \leq \mathsf{negl}(\kappa),$$

*then we say that $\Pi$ has computational unique responses, where* $\mathsf{negl}$ *is a negligible function.*

The proof of the following lemma is based on the same idea of [22, Lemma 6.7] and hence omitted. Here $G$ is the hash function used to generate challenges modeled as random oracle.

**Lemma 1.** *If the commitment scheme* $\mathsf{Com}$ *and the hash functions $H$ and $G$ used to implement* $\mathsf{Sig}[\Pi_{Res,2}]$ *are binding and collision-resistant, respectively, then* $\mathsf{Sig}[\Pi_{Res,2}]$ *has computation unique responses.*

## F Compressed one-out-of-$N$ proof for circuits

In this section, we present an improved one-out-of-$N$ proof for circuits by applying the parallel version of $\Pi_{Res,2}$ described in section 5.3 to the CDS method [26].

Consider a typical one-out-of-$N$ proof with $N$ statements $\{x_i\}_{i\in[N]}$ of a relation $R$, where the prover $P$ proves that he knows the witness $w$ for the relation $R_{\mathrm{OR}}$ defined below.

$$(x_1, \ldots, x_N \in L_R; w) \in R_{\mathrm{OR}} \Longleftrightarrow \exists t \in [N], s.t.(x_t, w) \in R.$$

[26] provides a novel method of the above disjunctive proof. Specifically, suppose there exists a 3-round public-coin HVZKPoK $\Pi_R$ for relation $R$. The protocol $\Pi_{\mathrm{OR}}$ for relation $R_{\mathrm{OR}}$ works as follows. The prover runs the simulator of $\Pi_R$ for each statement $x_i$ such that $i \neq t$ to generate the corresponding commitments, challenges and responses, e.g., $\{a_i, e_i, z_i\}_{i\neq t}$. For statement $x_t$, the prover runs $\Pi_R$ to generate the commitment $a_t$. Upon receiving the verifier's challenge $e$, the prover sets $e_t = e \oplus e_1 \oplus \cdots \oplus e_{t-1} \oplus e_{t+1} \oplus \cdots \oplus e_N$ and computes the responses $z_t$. The verifier checks the validity of all transcripts $\{a_i, e_i, z_i\}_{i\in[N]}$ and $e \overset{?}{=} e_1 \oplus \cdots \oplus e_N$.

By applying a parallel version of our $\Pi_{Res,2}$ to the above disjunctive proof for circuits $C$, we can get a compressed one-out-of-$N$ proof. In particular, suppose the $N$ statements are $\{x_i = (C, y_i)\}_{i \in [N]}$, where the prover has the knowledge of witness $w_t$ such that $C(w_t) = y_t$. The prover runs the CDS protocol using KKW proof, except the following main modifications.

1. Run the online phase of the MPC protocol for $C$ until it generates the intermediate masked inputs for $C'$.
2. Rerandomize the intermediate masked inputs for each statement. For the statement that the prover knows the witness, say $x_t$, run the resumable HVZKPoK $\Pi_{Res,2}$ of $C'$. For other $N-1$ statements, run the simulator of the resumable HVZKPoK $\Pi_{Res,2}$ of $C'$ in parallel such that the output of the simulated MPC protocol is the corresponding $y_i$.

More details of our compressed one-out-of-$N$ proof for circuits $\Pi_{\mathrm{OR}}$ are shown in Fig 6 and Fig 7.

**Remark**. We emphasize that the decomposition of $C$ in our compressed one-out-of-$N$ proof does not need to be extractable, since it has only one session and does not require resumable knowledge soundness. Therefore, $C$ can be decomposed such that the partial circuit $C'$ consists of the outputs wires of $C$ only. In this case, the only way to generate the consistent view with different $y_u$s is to modify the challenged party's share of the mask for $y_u$. Recall that the verifier needs to reconstruct the mask of $y_u$, say $\lambda^{(u)}$, using the shares of the related masks, say $[\lambda^{(u)}]_i$, and check that $y_u \overset{?}{=} \hat{y}_u \oplus \lambda^{(u)}$. Note that, for $u \neq t$, the challenged party, say $p^{(u)}$, is chosen by the prover in advance and the corresponding share $[\lambda^{(u)}]_{p^{(u)}}$ is broadcast as the message in step 7 of round 1. Hence, the prover can generate the consistent transcripts by modifying $[\lambda^{(u)}]_{p^{(u)}}$ such that $y_u = \hat{y}_u \oplus [\lambda^{(u)}]_1 \oplus \cdots \oplus [\lambda^{(u)}]_n$.

**Efficiency**. Although the complexity of our $\Pi_{\mathrm{OR}}$ is linear with the number of statements, the concrete computational and communicational costs of our $\Pi_{\mathrm{OR}}$ for $N$ statements can be very efficient when $N$ is small. It is just slightly higher than that of the KKW protocol for only one statement, since the prover only needs to run the simulation for the partial circuit $C'$ instead of $C$ for all but one statements.

**Theorem 5.** *Assume that the hash function $H$ is collision-resistant, and the commitment scheme* Com *and the pseudorandom generator PRG are secure. Then the protocol $\Pi_{\mathrm{OR}}$ is an honest verifier zero-knowledge proof of knowledge.*

*Proof.* **Completeness.** This property follows directly from the correctness of the underlying MPC protocol $\Pi$.

**Honest Verifier Zero-Knowledge.** Let $\mathsf{Sim}_\Pi$ denote the simulator of the MPC protocol $\Pi$. The description of simulator $\mathsf{Sim}_{\mathrm{OR}}$ for $\Pi_{\mathrm{OR}}$ is described as follows.

1. Choose random $\mathsf{salt}$ and $\mathsf{seed}_\Delta$, and generate $\mathsf{seed}_j^\Delta$ using $\mathsf{seed}_\Delta$ for each instance $j \in [M]$.
2. Choose uniform $\mathcal{C}$ and $\mathcal{P} = \{p_j\}_{j \in \mathcal{C}}$ as the challenge for the preprocessing phase and the online phase, respectively.

**Parameters:** Let $\kappa$ denote the computational security parameter. Let $H : \{0,1\}^* \to \{0,1\}^{2\kappa}$ be a collision-resistant hash function and $\mathsf{Com}$ be a secure commitment scheme. Let $\mathsf{PRG}$ be a secure pseudorandom generator. $M, n, \tau$ and $N$ denote the number of instances, the number of the parties, the number of instances whose preprocessing phase will not be opened and the number of the statements, respectively.
**Inputs:** Both the prover and the verifier have $N$ statements $(C, y_1), (C, y_2), \ldots, (C, y_N)$ and $C'$ as the partial circuit of $C$. The prover also holds a $w_t$ such that $C(w_t) = y_t$ for some $t \in [N]$.
**Round 1:** The prover chooses random $\mathsf{salt} \in \{0,1\}^\kappa$ and $\mathsf{seed}_\Delta \in \{0,1\}^\kappa$. For each $j \in [M]$, the prover proceeds as follows:

1. Compute $\mathsf{seed}_j^\Delta$ using $\mathsf{seed}_\Delta$ and choose uniform $\mathsf{seed}_j^* \in \{0,1\}^\kappa$ as the seed of $\mathsf{PRG}$ to generate
$(\mathsf{seed}_{j,1}, r_{j,1}), \ldots, (\mathsf{seed}_{j,n}, r_{j,n})$,
$(\mathsf{seed}_{j,1}^{(1)}, r_{j,1}^{(1)}), \ldots, (\mathsf{seed}_{j,n}^{(1)}, r_{j,n}^{(1)})$,
$\vdots$
$(\mathsf{seed}_{j,1}^{(N)}, r_{j,1}^{(N)}), \ldots, (\mathsf{seed}_{j,n}^{(N)}, r_{j,n}^{(N)})$.
2. Compute $\mathsf{aux}_{j,n} \in \{0,1\}^{|C|}$ as described in the text. For each $i \in [n-1]$, set $\mathsf{state}_{j,i} = \mathsf{seed}_{j,i}$; set $\mathsf{state}_{j,n} = \mathsf{seed}_{j,n} || \mathsf{aux}_{j,n}$.
3. For each $i \in [n]$, compute $\mathsf{com}_{j,i} = \mathsf{Com}(\mathsf{state}_{j,i}, r_{j,i}, \mathsf{salt})$.
4. Simulate the online phase of the MPC protocol for $C$ until it generates the masked intermediate input for $C'$, using $\{\mathsf{state}_{j,i}\}_{i\in[n]}$ and $\{\hat{z}_{j,\alpha}\}$, where $\{\hat{z}_{j,\alpha}\}$ denotes the masked inputs of $C$. Let $\mathsf{msgs}_{j,i}$ denote the messages broadcast by party $P_i$ during the protocol execution and $\{\hat{z}_{j,\alpha,inter}\}$ denote the corresponding masked intermediate value for each input wire $\alpha$ of $C'$.
5. Use $\mathsf{seed}_j^\Delta$ to generate $\{\Delta_{j,\alpha}^{(1)}\}, \ldots, \{\Delta_{j,\alpha}^{(N)}\}$ for each input wire $\alpha$ of $C'$. Then compute $\mathsf{aux}_{j,n}^{(u)} \in \{0,1\}^{|C'|+|C'_{in}|}$ for each $u \in [N]$ as described in the text. Set $\mathsf{state}_{j,n}^{(u)} = \mathsf{seed}_{j,n}^{(u)} || \mathsf{aux}_{j,n}^{(u)}$ and $\mathsf{state}_{j,i}^{(u)} = \mathsf{seed}_{j,i}^{(u)}$ for each $u \in [N]$ and $i \in [n-1]$.
6. For each $u \in [N]$ and $i \in [n]$, compute $\mathsf{com}_{j,i}^{(u)} = \mathsf{Com}(\mathsf{state}_{j,i}^{(u)}, r_{j,i}^{(u)}, \mathsf{salt})$.
7. For $u = t$, simulate the online phase of the MPC protocol for $C'$ using $\{\mathsf{state}_{j,i}^{(t)}\}_{i\in[n]}$ and $\{\hat{z}_{j,\alpha,inter} \oplus \Delta_{j,\alpha}^{(t)}\}$. Let $\mathsf{msgs}_{j,i}^{(t)}$ denote the messages broadcast by party $P_i$ during the protocol execution.
For each $u \in [N]$ such that $u \neq t$, choose a random challenge $p_j^{(u)}$ and run $\mathsf{Sim}_\Pi$ to simulate the online phase of the MPC protocol for $C'$ using $\{\mathsf{state}_{j,i}^{(u)}\}_{i \neq p_j^{(u)}}$ and $\{\hat{z}_{j,\alpha,inter} \oplus \Delta_{j,\alpha}^{(u)}\}$ so that the related views are consistent with $y_u$. Here, the message broadcast by party $P_i$ during the simulation are denoted as $\mathsf{msgs}_{j,i}^{(u)}$, which are output by $\mathsf{Sim}_\Pi$.
8. Let $h_j = H(\mathsf{com}_{j,1}, \ldots, \mathsf{com}_{j,n}, \mathsf{com}_{j,1}^{(1)}, \ldots, \mathsf{com}_{j,n}^{(1)}, \ldots, \mathsf{com}_{j,1}^{(N)}, \ldots, \mathsf{com}_{j,n}^{(N)})$ and $h'_j = H(\{\hat{z}_{j,\alpha}\}, \mathsf{seed}_j^\Delta, \mathsf{msgs}_{j,1}, \ldots, \mathsf{msgs}_{j,n}, \mathsf{msgs}_{j,1}^{(1)}, \ldots, \mathsf{msgs}_{j,n}^{(1)}, \ldots, \mathsf{msgs}_{j,1}^{(N)}, \ldots,$ $\mathsf{msgs}_{j,n}^{(N)}, \mathsf{salt})$.

Compute $h = H(h_1, \ldots, h_M)$, $h' = H(h'_1, \ldots, h'_M)$ and $h^* = H(h, h')$. Send $h^*$ to the verifier.

**Fig. 6.** Compressed one-out-of-$N$ proof for circuits $\Pi_{\mathrm{OR}}$ (part 1)

**Round 2:** The verifier chooses a uniform $\tau$-sized set $\mathcal{C} \subset [M]$ and $\mathcal{P} = \{p_j\}_{j \in \mathcal{C}}$, where each $p_j \in [n]$ is uniform. Send $(\mathcal{C}, \mathcal{P})$ to the prover.
**Round 3:** The prover's response includes:

1. $\mathsf{salt}$ and $\mathsf{seed}_\Delta$.
2. $\mathsf{seed}_j^*$ and $h'_j$, for each $j \in [M] \backslash \mathcal{C}$.
3. $\{\mathsf{state}_{j,i}, r_{j,i}\}_{i \neq p_j}$, $\mathsf{msgs}_{j,p_j}$, $\mathsf{com}_{j,p_j}$, $\{\hat{z}_{j,\alpha}\}$ and $\mathcal{P}_j = \{p_j^{(u)}\}_{u \in [N]}$, for each $j \in \mathcal{C}$, where $p_j^{(t)} = p_j \oplus p_j^{(1)} \oplus \cdots \oplus p_j^{(t-1)} \oplus p_j^{(t+1)} \oplus \cdots \oplus p_j^{(N)}$.
4. $\{\mathsf{state}_{j,i}^{(u)}, r_{j,i}^{(u)}\}_{i \neq p_j^{(u)}}$, $\mathsf{msgs}_{j,p_j^{(u)}}^{(u)}$ and $\mathsf{com}_{j,p_j^{(u)}}^{(u)}$, for each $j \in \mathcal{C}$ and $u \in [N]$.

**Verification:** The verifier accepts iff all the following checks are satisfied.

1. For each $j \in [M]$, compute $\mathsf{seed}_j^\Delta$ using $\mathsf{seed}_\Delta$.
2. For each $j \in \mathcal{C}$,
   i. Check that $p_j = p_j^{(1)} \oplus \cdots \oplus p_j^{(N)}$.
   ii. Use $\{\mathsf{state}_{j,i}, r_{j,i}\}_{i \neq p_j}$ and $\mathsf{salt}$ to compute $\{\mathsf{com}_{j,i}\}_{i \neq p_j}$.
   iii. For each $u \in [N]$, use $\{\mathsf{state}_{j,i}^{(u)}, r_{j,i}^{(u)}\}_{i \neq p_j^{(u)}}$ and $\mathsf{salt}$ to compute $\{\mathsf{com}_{j,i}^{(u)}\}_{i \neq p_j^{(u)}}$. Then compute $h_j = H(\mathsf{com}_{j,1}, \ldots, \mathsf{com}_{j,n}, \mathsf{com}_{j,1}^{(1)}, \ldots, \mathsf{com}_{j,n}^{(1)}, \ldots, \mathsf{com}_{j,1}^{(N)}, \ldots, \mathsf{com}_{j,n}^{(N)})$.
3. For each $j \in [M] \backslash \mathcal{C}$, use $\mathsf{seed}_j^*$, $\mathsf{seed}_j^\Delta$ and $\mathsf{salt}$ to compute $h_j$ as an honest prover would. Then compute $h = H(h_1, \ldots, h_M)$.
4. For each $j \in \mathcal{C}$,
   i. Using $\{\mathsf{state}_{j,i}, r_{j,i}\}_{i \neq p_j}$, $\{\hat{z}_{j,\alpha}\}$ and $\mathsf{msgs}_{j,p_j}$, simulate the online phase of the MPC protocol for $C$ until it generates the intermediate masked-inputs for $C'$. This yields $\{\mathsf{msgs}_{j,i}\}_{i \neq p_j}$ and the intermediate masked-inputs $\{\hat{z}_{j,\alpha,inter}\}$.
   ii. For each $u \in [N]$, simulate the online phase of the MPC protocol for $C'$ using $\{\mathsf{state}_{j,i}^{(u)}, r_{j,i}^{(u)}\}_{i \neq p_j^{(u)}}$, the rerandomized intermediate masked inputs $\{\hat{z}_{j,\alpha,inter} \oplus \Delta_{j,\alpha}^{(u)}\}$ and $\mathsf{msgs}_{j,p_j^{(u)}}^{(u)}$. This yields $\{\mathsf{msgs}_{j,i}^{(u)}\}_{i \neq p_j^{(u)}}$ and the reconstruction of $y'_u$. Check that $y_u \overset{?}{=} y'_u$ and computes $h'_j = H(\{\hat{z}_{j,\alpha}\}, \mathsf{seed}_j^\Delta, \mathsf{msgs}_{j,1}, \ldots, \mathsf{msgs}_{j,n}, \mathsf{msgs}_{j,1}^{(1)}, \ldots, \mathsf{msgs}_{j,n}^{(1)}, \ldots, \mathsf{msgs}_{j,1}^{(N)}, \ldots, \mathsf{msgs}_{j,n}^{(N)}, \mathsf{salt})$.

   Compute $h' = H(h'_1, \ldots, h'_M)$.
5. Check that $H(h, h') \overset{?}{=} h^*$.

**Fig. 7.** Compressed one-out-of-$N$ proof for circuits $\Pi_{\mathrm{OR}}$ (part 2)

3. For each instance $j$ not in $\mathcal{C}$, choose uniform $\mathsf{seed}^*_j$ and generate $\mathsf{seed}^{\Delta}_j$ using $\mathsf{seed}_{\Delta}$. Then, generate $h_j$ as an honest prover would and compute $h'_j$ as a hash value to a random string.
4. For each instance $j$ in $\mathcal{C}$, run $\mathsf{Sim}_{\Pi}$ to simulate the views of $n$ parties in the execution of the MPC protocol $\Pi$ for $C$ until it can generate the masked intermediate inputs for $C'$. This yields $\{\mathsf{state}_{j,i}\}_{i \neq p_j}$, masked-input $\{\hat{z}_{j,\alpha}\}$ for $C'$, $\mathsf{msgs}_{j,p_j}$ and corresponding intermediate masked-input $\{\hat{z}_{j,\alpha,inter}\}$. Compute $\mathsf{com}_{j,i}$ for $i \neq p_j$ as an honest prover and compute $\mathsf{com}_{j,p_j}$ as a commitment to a 0-string.
5. For each instance $j$ in $\mathcal{C}$, choose uniform $\{p^{(u)}_j\}_{u \neq t}$ and compute $p^{(t)}_j = p_j \oplus p^{(1)}_j \oplus \cdots \oplus p^{(t-1)}_j \oplus p^{(t+1)}_j \oplus \cdots \oplus p^{(N)}_j$.
6. For each instance $j$ in $\mathcal{C}$, run $\mathsf{Sim}_{\Pi}$ to simulate the views of $n$ parties in the execution of the MPC protocol $\Pi$ for $C'$ with the rerandomized masked inputs $\{\hat{z}_{j,\alpha,inter} \oplus \Delta^{(u)}_{j,\alpha}\}$ (where $\Delta^{(u)}_{j,\alpha}$ is generated by $\mathsf{seed}^{\Delta}_j$) for each $u \in [N]$, which yields $\{\mathsf{state}^{(u)}_{j,i}\}_{i \neq p^{(u)}_j}$ and $\mathsf{msgs}^{(u)}_{j,p^{(u)}_j}$. Choose random $r^{(u)}_{j,i}$ to compute $\mathsf{com}^{(u)}_{j,i}$ for $i \neq p^{(u)}_j$ and compute $\mathsf{com}^{(u)}_{j,p^{(u)}_j}$ as a commitment to a 0-string. Then compute $h_j$ and $h'_j$ as an honest prover would.
7. Compute $h^*$ as an honest prover would and output the simulated transcripts, which includes:
   - $h^*, \mathsf{salt}, \mathsf{seed}_{\Delta}, \mathcal{C}, \mathcal{P}$;
   - For each $j \in [M] \backslash \mathcal{C}$, $\mathsf{seed}^*_j, h'_j$;
   - For each $j \in \mathcal{C}$, $\{\mathsf{state}_{j,i}, r_{j,i}\}, \mathsf{msgs}_{j,p_j}, \mathsf{com}_{j,p_j}, \{\hat{z}_{j,\alpha}\}, \mathcal{P}_j$.
   - For each $j \in \mathcal{C}, u \in [N]$, $\{\mathsf{state}^{(u)}_{j,i}, r^{(u)}_{j,i}\}_{i \neq p^{(u)}_j}, \mathsf{msgs}^{(u)}_{j,p^{(u)}_j}, \mathsf{com}^{(u)}_{j,p^{(u)}_j}$.

Following a standard hybrid argument, the transcripts generated by $\mathsf{Sim}_{\mathrm{OR}}$ are computationally indistinguishable from that of a real protocol, where the indistinguishability relies on the indistinguishability of the simulated transcripts generated by $\mathsf{Sim}_{\Pi}$ and the hiding property of the commitment scheme.

**Knowledge Soundness.** We need to show that there exists a witness extractor $\mathcal{E}_{\mathrm{OR}}$ which can extract a valid witness if the success probability higher than the soundness error. Since $\{p^{(u)}_j\}_{u \neq t}$ is chosen by the prover and $p^{(t)}_j$ is computed by $p^{(t)}_j = p_j \oplus p^{(1)}_j \oplus \cdots \oplus p^{(t-1)}_j \oplus p^{(t+1)}_j \oplus \cdots \oplus p^{(N)}_j$ where $p_j$ is chosen by the verifier, a malicious prover who succeeds in cheating in the preprocessing phase or guessing the challenge of the online phase of the MPC protocol for $C'$ is able to convince the verifier to accept the proof. Thus, the soundness error $\xi_{\mathrm{OR}}(M, n, \tau)$ of $\mathit{\Pi}_{\mathrm{OR}}$ is the same as $\xi_1(M, n, \tau)$ of $\mathit{\Pi}_{Res,1}$. Similarly, using the method of Theorem 2, we can prove that if the success probability $\delta_{\mathrm{OR}}(x) > \xi_{\mathrm{OR}}(M, n, \tau)$, there exists a witness extractor $\mathcal{E}_{\mathrm{OR}}$ which can extract a valid witness in the expected number of steps bounded by $O(\frac{1}{\delta_{\mathrm{OR}}(x) - \xi_{\mathrm{OR}}(M,n,\tau)})$.

## G Ring Signature from Symmetric-Key Primitives

A ring signature scheme [56] allows a member of an ad-hoc group (or ring) to sign a message on behalf of the group anonymously. It is known that the 1-out-of-$N$

proofs [26] together with the Fiat-Shamir heuristic provides a straightforward construction of ring signatures [5]. In particular, the prover proves that he knows the secret key $sk$ of some public key in the ring. So the relation $R_{\text{OR}}$ is defined as

$$(pk_1, \ldots, pk_N \in L_R; sk) \in R_{\text{OR}} \Longleftrightarrow \exists t \in [N], s.t.(pk_t, sk) \in R,$$

where $(pk_t, sk) \in R$ is defined by $pk_t = F(sk)$ for a one-way function $F$. One drawback of such construction is the size of the resulting ring signature is linear with the size of the ring. To reduce the signature size, the Merkle-tree based accumulator is applied to the disjunctive proof [31], which results in logarithm-size ring signatures. To the best of our knowledge, all the efficient constructions of ring signatures from symmetric-key primitives rely on the Merkle-tree based accumulator [51,28,15].

We find that, by replacing the 1-out-of-$N$ proofs [26] with our compressed 1-out-of-$N$ proofs in the straightforward construction, the resulting ring signature (without using accumulator) can achieve better concrete efficiency than that of the state-of-the-art construction [51] when the size of the ring is small. More precisely, the cost of our ring signature is comparable with that of one KKW signature [51] plus the following overheads: $\{\mathsf{seed}_\Delta;\ \mathcal{P}_j$, for each $j \in \mathcal{C}$; $\{\mathsf{state}_{j,i}^{(u)}, r_{j,i}^{(u)}\}_{i \neq p_j^{(u)}}$, $\mathsf{msgs}_{j,p_j^{(u)}}^{(u)}$ and $\mathsf{com}_{j,p_j^{(u)}}^{(u)}$, for each $j \in \mathcal{C}$ and $u \in [N]\}$. The aforementioned overheads have the communication complexity at most

$$\kappa + \tau \cdot (N \cdot \log n + N \cdot (\kappa \cdot \log n + |C'| + |C'_{in}|) + N \cdot 2\kappa + (N-1) \cdot (|C'| + |C'_{out}|)).$$

Note that $|C'|$ can be 0, since $C'$ can consist of the the output wires of $C$ only as explained in Appendix F.