Functional Encryption Supporting Recursive Languages

Somindu C. Ramanna
Indian Statistical Institute, Kolkata
somindu_r@isical.ac.in

Palash Sarkar
Indian Statistical Institute, Kolkata
palash@isical.ac.in

Abstract

We provide a construction for functional encryption over the set of recursive languages. In this scheme, a secret key $\mathcal{SK}_{\mathcal{M}}$ encodes a halting double-stack deterministic pushdown automaton (2DPDA) $\mathcal{M}$ that accepts by final state. Encryption algorithm takes a message m and a string w as input and outputs a ciphertext $\mathcal{C}$. A user possessing $\mathcal{SK}_{\mathcal{M}}$ can decrypt $\mathcal{C}$ only if $\mathcal{M}$ accepts w. Halting 2DPDAs can simulate halting deterministic Turing machines and hence our construction essentially covers all recursive languages.

The construction is built upon Waters' bilinear pairing-based functional encryption scheme over regular languages. The main technical novelty is in handling stack contents and λ-transitions (i.e., transitions that do not advance the input pointer) of the automata. This is reflected both in the construction and the security arguments. The scheme is shown to be selectively secure based on the decision ℓ-expanded bilinear Diffie-Hellman exponent assumption introduced by Waters.

1 Introduction

Functional encryption is a new form of public key encryption that provides sophisticated access control based on certain policies and also ability to compute functions over encrypted data. In a functional encryption (FE) system, a sender encrypts a message x to a ciphertext $\mathcal{C} = \mathsf{Encrypt}(\mathcal{PP}, x)$, where $\mathcal{PP}$ denotes the public parameters of the system. A trusted authority called the private key generator (PKG) issues secret keys to users. A secret key $\mathcal{SK}_f$ enables computation of the function $f(\cdot)$. Decryption algorithm computes $f(x)$ given the encryption $\mathcal{C}$ of x. The decrypting entity cannot learn anything more than $f(x)$ about x. A formal treatment of functional encryption, its definition and security models can be found in [BSW12].

In systems requiring access control, the plaintext x is of the form (m, Ψ) where m is the message and Ψ is an *index* that describes a user's credentials. The access policy would be defined via a predicate. The function $f(\cdot)$ takes in a ciphertext encrypting (m, Ψ) and returns m if the predicate evaluates to true on input Ψ; and $\perp$ otherwise. In this work we only look at the *public index* model, where the index Ψ is not hidden. The form of FE described above is called *key-policy* functional encryption since the policy is encoded in the key. A complementary form called *ciphertext-policy* FE is also studied where the function is hard coded in the ciphertext and index in the key.

Functional encryption schemes with different kinds of functionalities have been studied. These include attribute-based encryption [SW05, GPSW06, OSW07, BSW07, Wat11], inner-product encryption [KSW08, OT09, OT10, LOS$^+$10] and many others in both the ciphertext-policy and key-policy settings. All of these schemes have one property in common – the functions only deal with fixed-size inputs. Waters [Wat12] went beyond the realm of fixed-size inputs and proposed a functional encryption scheme that operates over arbitrary sized inputs. A secret key is associated with a deterministic finite automaton (DFA) $\mathcal{M}$ and the index Ψ is a string w over the input alphabet of the DFA. Decryption succeeds if $\mathcal{M}$ accepts w. The set of langauges the system can deal with is restricted to the class of regular languages. A natural question put forward in [Wat12] is whether it is possible to obtain a construction which works with a Turing Machine (TM) instead of a DFA?

Our Contributions. We construct a functional encryption scheme over *recursive languages*. A language is recursive if there is a halting TM which accepts it. It has been indicated in [Wat12] that achieving this is difficult since one approach to building FE based on Turing machines requires the ability to tackle non-determinism; and handling non-determinism is not straightforward due to backtracking attacks (described in [Wat12]). On the other hand, it is well-known that at the top of the language hierarchy, both determinism and non-determinism have the same power. Keeping this in mind, we move up the hierachy and obtain a scheme which works with deterministic Turing machines without having to tackle with non-determinism at all and also the resulting backtracking attacks. A two-step approach was followed.

- First, the scheme of Waters [Wat12] was extended to work over deterministic pushdown automata (DPDA). The main challenge was to devise ways to handle the stack contents and λ-transitions (here λ is the empty string and the transition does not read any input symbol). The resulting scheme is secure in the selective model under the expanded ℓ-BDHE assumption that was introduced by Waters in [Wat12] to prove security of the DFA-based FE scheme.
- The next step was to consider automata equipped with two stacks. A two-stack determinitic push-down automaton can simulate a deterministic Turing machine and hence we are done. Moving from one stack to two stacks was relatively easy.

While our construction can handle any recursive language, the proof only holds for languages recognisable in polynomial time (i.e., all recursive languages in the class $\mathscr{P}$). The reason is that the attacker does not have unbounded resources. More specifically, the adversary is a probabilistic algorithm running in time bounded by a polynomial in the security parameter. There could be automata that take exponential time (in security parameter) to decide and halt. Allowing such automata in the proof provides exponential time to the adversary for attacking the scheme. Therefore, within this framework, it is not possible to obtain a meaningful proof covering the set of all recursive languages.

We only present the more general two-stack version in Section 4 along with a security proof in the selective identity model. Definitions of DPDA-based functional encryption and the corresponding security model are provided in Section 3 along with some background on pushdown automata and langauges.

Related Work. A special case of functional encryption is identity-based encryption [BF03, Coc01] where the function is just an equality test for the identities corresponding to the secret key and the ciphertext. More complex access policies can be supported by more sophisticated functionalities. For example, in an attribute-based encryption system [SW05, GPSW06], a set of *attributes* define users' credentials and the function encodes an access struture which is nothing but a boolean formula over these attributes. Other classes of functional encryption include spatial encryption [Ham11] and inner-product encryption [OT09, OT10, KSW08]. More generally, Sahai and Waters [SW12] constructed an attribute-based encryption scheme for general circuits using multi-linear maps [GGH12]. Further, Goldwasser *et.al.* [GKP$^+$12] addresses the problem of constructing functional encryption for any general function by combining a public index predicate encryption scheme with homomorphic encryption. None of these works seem to subsume the result in Waters [Wat12] and by extension neither do they subsume the result of the current work.

2 Pushdown Automata

We start with a brief overview of pushdown automata, double-stack machines and languages.

Definition 2.1. A pushdown automaton (PDA) $\mathcal{M}$ is a 7-tuple $(Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$ where

- $Q \neq \emptyset$ is a finite set of *states*,
- $\Sigma \neq \emptyset$ denotes the *input alphabet*,
- $\Gamma \neq \emptyset$ is the *stack alphabet*,
- $q_0 \in Q$ is the *start state*,
- $Z_0 \in \Gamma$ is a special symbol marking *bottom of the stack*,
- $\emptyset \neq F \subseteq Q$ is the set of *final states* and
- $\delta : Q \times \Sigma \cup \{\lambda\} \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$ is called the *transition function*.

An *instantaneous description* (ID) describes the configuration of a PDA (state and stack contents) at a given instant. An ID is a triple (q, w, γ) where q is a state, w and γ are strings over Σ and Γ respectively. Let $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$ be a PDA, $a \in \Sigma \cup \{\lambda\}$, $w \in \Sigma^*$, $q, q' \in Q$, $Z \in \Gamma$ and $\gamma, \beta \in \Gamma^*$. Here λ denotes the *empty string*. We write $(q, aw, Z\gamma) \vdash_{\mathcal{M}} (q', w, \beta\gamma)$ if $(q', \beta) \in \delta(q, a, Z)$. The notation $\vdash^*_{\mathcal{M}}$ denotes the reflexive and transitive closure of $\mathcal{M}$. We say that $\mathcal{M}$ *accepts a string* $w \in \Sigma^*$ if $(q_0, w, Z_0) \vdash^*_{\mathcal{M}} (q_x, \lambda, \gamma)$ for some $q_x \in F$ and $\gamma \in \Gamma^*$. In other words, $\mathcal{M}$ accepts w if there is sequence of transitions so that on reading the string w, $\mathcal{M}$ enters a final state. This notion of acceptance is termed *acceptance by final state*. Another definition used is that of *acceptance by empty stack* in which an automaton $\mathcal{M}$ accepts string w if $(q_0, w, Z_0) \vdash^*_{\mathcal{M}} (q_x, \lambda, \lambda)$ for some $q_x \in Q$. In this context, the set of final states F is the empty set.

Given a PDA $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$, define the *language accepted by* $\mathcal{M}$ *by final state* as

$$\mathcal{L}(\mathcal{M}) = \{w \in \Sigma^* : \mathcal{M} \text{ accepts } w \text{ by final state}\},$$

and similarly the *language accepted by* $\mathcal{M}$ *by empty stack* to be

$$\mathcal{N}(\mathcal{M}) = \{w \in \Sigma^* : \mathcal{M} \text{ accepts } w \text{ by empty stack}\}.$$

The two notions of acceptance are equivalent and hence for any PDA $\mathcal{M}$, $\mathcal{L}(\mathcal{M}) = \mathcal{N}(\mathcal{M}')$ for some PDA $\mathcal{M}'$. The class of languages accepted by PDAs $\{L \subseteq \Sigma^* : \text{there exists a PDA } \mathcal{M} \text{ such that } L = \mathcal{L}(\mathcal{M})\}$ is precisely the set of all *context-free languages* (CFLs).

Determinism. A pushdown automaton $\mathcal{M}$ is said to be *deterministic* if it satisfies the following two conditions.

- For any $q \in Q$, $\sigma \in \Sigma \cup \{\lambda\}$ and $Z \in \Gamma$, the set $\delta(q, \sigma, Z)$ has at most one element.
- For any $q \in Q$, $Z \in \Gamma$, if $\delta(q, \lambda, Z) \neq \emptyset$ then $\delta(q, \sigma, Z) = \emptyset$ for every $\sigma \in \Sigma$.

Acceptance by a DPDA can be defined in two ways as discussed earlier, but the two notions are not equivalent. The class of languages accepted by final state are called *deterministic context-free languages* (DCFLs). Languages accepted by empty stack are DCFLs with the additional property that no string in a language is a prefix of another string in the language.

2-Stack Machines. A 2-stack PDA (2PDA) $\mathcal{M}$ is an 8-tuple $(Q, \Sigma, \Gamma_1, \Gamma_2, q_0, Z_0^{(1)}, Z_0^{(2)}, F, \delta)$ where all the parameters are defined as for the single stack PDA with the following differences.

- Γ_1, Γ_2 are the alphabets for stack 1 and 2 respectively.
- The bottom-of-stack markers for the two stacks are $Z_0^{(1)} \in \Gamma_1$ and $Z_0^{(2)} \in \Gamma_2$.
- The transition function is now of the form $\delta : Q \times \Sigma \cup \{\lambda\} \times \Gamma_1 \times \Gamma_1 \to \mathcal{P}(Q \times \Gamma_1^* \times \Gamma_2^*)$

An ID for a 2-stack PDA is a quadruple $(q, w, \gamma_1, \gamma_2)$ where q is a state, w is the input string to be read and γ_1, γ_2 are strings over Γ_1, Γ_2. We write $(q, aw, Z_1\gamma_1, Z_2\gamma_2) \vdash_{\mathcal{M}} (q', w, \beta_1\gamma_1, \beta_2\gamma_2)$ if $(q', \beta_1, \beta_2) \in \delta(q, a, Z_1, Z_2)$ (here $\beta_1 \in \Gamma_1^*, \beta_2 \in \Gamma_2^*$). The two stacks may have the same stack alphabet. In the rest of the paper, we use the same alphabet for both the stacks i.e., $\Gamma_1 = \Gamma_2 = \Gamma$ and set $Z_0^{(1)} = Z_0^{(2)} = Z_0$. Also the two stacks could have two different symbols from Γ as the bottom-of-stack markers. Determinism and acceptance are defined in a manner similar to the single-stack PDAs.

More generally, one can define the notion of k-stack PDA in a manner similar to that of 2-stack PDA.

Turing Machines (TMs). The most powerful model of computation is that of a TM. Here, we do not formally define TMs since we will not require the definition and instead refer to [HMU00] for the definition. The class of languages accepted by TMs is the class of *recursively enumerable (r.e.)* languages. It is well known that for TM, non-determinism does not provide addtional power, i.e., for any r.e. language there is a deterministic TM (DTM) which accepts it. So, we will only consider DTMs. A language is said to be *recursive* if there is a DTM which accepts the language and *always* halts. The class of *recursive* languages is a proper subset of the class of r.e. languages.

TMs and PDAs. It is known that a TM can be simulated by a 2-stack PDAs [HMU00, Chapter 8]. Essentially the tape is simulated by two stacks with the contents to the left of the head on one stack and the contents to the right of the head on the other stack. The connection between TMs and 2-stack PDAs extends to the deterministic versions: Given a deterministic TM (DTM) it is easy to construct a 2-stack deterministic PDA (DPDA) which simulates the DTM. In view of this, the class of languages accepted by 2-stack DPDAs is also the class of r.e. languages.

Halting 2-stack DPDAs. The most powerful (or expressive) form would be to work with arbitrary 2-stack DPDAs. This, however, causes a problem. On certain inputs, a given machine may never halt. This would mean that the decryption algorithm of the functional encryption scheme does not terminate. To ensure that the decryption algorithm always terminates we require the 2-stack DPDA to halt on all inputs. So, halting 2-stack DPDAs accept the class of recursive languages.

Issue about Halting Time. In the security proof, we require the DPDAs that are queried during the key extraction phase to halt and decide in polynomial time (in the security parameter) due to the following reason. The adversary is modeled as a probabilistic t-time algorithm where t is a polynomial in the security parameter. The reduction exploits the adversary's advantage in breaking the scheme in solving some *computationally hard* problem. If the DPDAs for which the adversary requests keys do not halt and decide in polynomial time then the simulator (or the algorithm interacting with the attacker to solve the hard problem) itself runs does not run in polynomial time since it has to check whether the challenge string is accepted by the queried automaton or not. This allows more time to the adversary which could eventually break the scheme.

Hence the construction works for any recursive language but the proof only holds for functional encryption schemes supporting polynomial-time-recognisable recursive languages. In fact, it is not possible to prove security for all recursive languages based on any computational assumption.

Reading the entire input. A halting 2-stack DPDA may accept an input before reading it completely. This causes problems in the construction of the decryption algorithm. Hence, we require the 2-stack DPDA to read the full input before halting and deciding whether to accept or reject.

In the sequel, we write 2DPDA to denote "a 2-stack DPDA that reads the entire input, halts and accepts by final state". We provide construction for FE over 2DPDA.

3 Definitions and Notation

Provided here are some notation and definitions of DPDA-based functional encryption followed by the security model.

3.1 Notation.

The following notation is used for any 2DPDA $\mathcal{M}$.

$$\mathsf{Accept}(\mathcal{M}, w) = \begin{cases} 1 & \text{if } \mathcal{M} \text{ accepts } w; \\ 0 & \text{otherwise.} \end{cases}$$

Transitions of an automaton $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, Z_0, Z_0, F, \delta)$ are represented as 5-tuples of the form $t = (q_x, q_{x'}, \sigma, Z_{y_1}, Z_{y_2}, Z_{y'_1}, Z_{y'_2})$ where $\delta(q_x, \sigma, Z_{y_1}, Z_{y_2}) = \{(q_{x'}, \gamma_1, \gamma_2)\}$ for some $\gamma_1, \gamma_2 \in \Gamma^*$ and $Z_{y'_1}$,$Z_{y'_2}$ are the elements at the top of the two stacks after the transition. $\mathcal{T}$ denotes the set of all transition tuples t.

The notation $[a, b]$ represents the set $\{x : a \le x \le b\}$ for two integers $a < b$. For a set $\mathcal{X}$, the notation $x_1, \ldots, x_k \in_{\mathrm{R}} \mathcal{X}$ indicates that $x_1, \ldots, x_k$ are independent random elements of $\mathcal{X}$ with some distribution R and $x_1, \ldots, x_k \xleftarrow{\mathrm{R}} \mathcal{X}$ symbolises $x_1, \ldots, x_k$ being sampled independently from $\mathcal{X}$ according to R. The uniform distribution is denoted by U. For a (probabilistic) algorithm $\mathcal{A}$, $x \longleftarrow \mathcal{A}(\cdot)$ means that x is chosen according to the output distribution of $\mathcal{A}$ (which of course may be determined by its input).

A (symmetric) pairing is represented as a tuple $(p, \mathbb{G}, \mathbb{G}_T, e, P)$ where p is a prime, $|\mathbb{G}| = |\mathbb{G}_T| = p$, $\mathbb{G} = \langle P \rangle$ and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is the pairing function.

3.2 Functional Encryption based on 2DPDAs

We provide a definition of functional encryption specific to 2DPDAs rather than following the general definition given in [BSW12]. A functional encryption (FE) scheme over 2DPDAs consists of four probabilistic algorithms - Setup, KeyGen, $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$.

- Setup: takes as input a security parameter κ, generates the public parameters $\mathcal{PP}$ and the master secret $\mathcal{MSK}$ based on λ and the input alphabet Σ. Σ is part of $\mathcal{PP}$.

- KeyGen: receives the description of a 2DPDA $\mathcal{M}$ and master secret $\mathcal{MSK}$ and outputs a secret key $\mathcal{SK}_{\mathcal{M}}$ corresponding to $\mathcal{M}$.
- Encrypt: inputs a message m, a string $w = w_1 w_2 \cdots w_\ell$ over Σ and returns a ciphertext $\mathcal{C}$ (which also contains w).
- Decrypt: inputs a ciphertext $\mathcal{C}$ and secret key $\mathcal{SK}_{\mathcal{M}}$. If $\mathsf{Accept}(\mathcal{M}, w) = 1$, the algorithm returns m; otherwise, returns $\perp$ indicating failure.

3.3 Security

Security proofs for our schemes are based on the notion of indistinguishability of ciphertexts under a chosen plaintext attack (CPA). It is defined via a game ind-cpa between an adversary $\mathscr{A}$ and a challenger consisting of several stages.

Setup: The challenger runs the Setup algorithm of the FE scheme and gives the public parameters to $\mathscr{A}$.

Phase 1: $\mathscr{A}$ makes a number of key extraction queries adaptively. For a query on automaton $\mathcal{M}$, the challenger runs the KeyGen algorithm of the FE scheme and returns its output $\mathcal{SK}_{\mathcal{M}}$ to $\mathscr{A}$.

Challenge: $\mathscr{A}$ provides two messages pairs m_0, m_1 and a challenge string $w^* = w_1^* w_2^* \cdots w_\ell^*$ subject to the condition that $\mathscr{A}$ does not request keys for any automaton that accepts w^* in **Phase 1** or **Phase 2**. The challenger then picks $\beta \xleftarrow{\text{U}} \{0, 1\}$ and returns an encryption $\mathcal{C}^*$ of m_β under the string w^* to $\mathscr{A}$.

Phase 2: $\mathscr{A}$ issues more key extraction queries as in **Phase 1** with the restriction that none of the automata that are queried accept w^*.

Guess: $\mathscr{A}$ outputs a bit β'.

In the selective model, there is a stage **Initialise** before **Setup** in which the adversary commits to the input alphabet Σ and the challenge string w^*. Call this game ind-s-cpa.

If $\beta = \beta'$, then $\mathscr{A}$ wins the game. The advantage of $\mathscr{A}$ in breaking the security of the FE scheme in the ind-cpa game is given by

$$\mathsf{Adv}_{\mathrm{FE}}^{\mathsf{ind\text{-}cpa}}(\mathscr{A}) = \left| \Pr[\beta = \beta'] - \frac{1}{2} \right|.$$

The FE scheme is said to be (ε, t, ν)-IND-STR-CPA secure[1] (secure under chosen plaintext attack) if for every adversary $\mathscr{A}$ making at most ν queries and whose running time is t, it holds that $\mathsf{Adv}_{\mathrm{FE}}^{\mathsf{IND\text{-}STR\text{-}CPA}}(\mathscr{A}) \leq \varepsilon$.

Security in the selective model (IND-sSTR-CPA-security) is defined in a similar manner with respect to ind-s-cpa game.

4 Realizing FE over Recursive Languages

A construction for functional encryption over 2-stack deterministic pushdown automata (2DPDA) is provided here. Security is proved using the expanded ℓ-BDHE assumption.

[1]The abbreviation "STR" stands for string. "sSTR" denotes that the challenge string is chosen selectively.

4.1 Construction

$\mathsf{Setup}(\Sigma, \kappa)$: Generate a symmetric pairing $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, e, P)$ according to the security parameter κ. Choose elements $P_1, H_{\text{start}}, H_{\text{end}}, H_\lambda, \{H_\sigma\}_{\sigma\in\Sigma} \xleftarrow{\text{U}} \mathbb{G}$ and $\alpha \xleftarrow{\text{U}} \mathbb{Z}_p$. The public parameters and master secret are given by

$$\begin{aligned}&\mathcal{PP}\ \ : (\mathcal{G}, \Sigma, P_1, H_{\text{start}}, H_{\text{end}}, H_\lambda, \{H_\sigma\}_{\sigma\in\Sigma}, e(P,P)^\alpha),\\ &\mathcal{MSK}\text{: } -\alpha P.\end{aligned}$$

$\mathsf{Encrypt}(\mathcal{PP}, w = w_1 \cdots w_\ell, m)$: Choose randomizers $s_0, s_1, \ldots, s_\ell \xleftarrow{\text{U}} \mathbb{Z}_p$. Compute the ciphertext elements as follows.

$$\begin{aligned}&C_m = m \cdot e(P,P)^{\alpha s_\ell},\\ &C_{0,1} = C_{\text{start},1} = s_0 P, \quad C_{\text{start},2} = s_0 H_{\text{start}},\\ &C_{i,1} = s_i P, \quad C_{i,2} = s_i H_{w_i} + s_{i-1} P_1, \quad C_{i,3} = s_{i-1} H_\lambda + s_{i-1} P_1 \quad \text{ for } i = 1, \ldots, \ell,\\ &C_{\text{end},1} = s_\ell P, \quad C_{\text{end},2} = s_\ell H_{\text{end}}.\end{aligned}$$

The ciphertext is given by $\mathcal{C} = (C_m, C_{\text{start},1}, C_{\text{start},2}, \{C_{i,1}, C_{i,2}, C_{i,3}\}_{i\in[1,\ell]}, C_{\text{end},1}, C_{\text{end},2}, w)$.

$\mathsf{KeyGen}(\mathcal{MSK}, \mathcal{M} = (Q, \Sigma, \Gamma, q_0, Z_0, Z_0, F, \delta))$: For each $(x, y_1, y_2) \in \mathbb{Z}_{|Q|} \times \mathbb{Z}_{|\Gamma|} \times \mathbb{Z}_{|\Gamma|}$, pick $D_{x,y_1,y_2} \xleftarrow{\text{U}} \mathbb{G}$. Also, choose elements r_{start}, for all $t \in \mathcal{T}$, r_t and for all $q_x \in F, Z_{y_1}, Z_{y_2} \in \Gamma$, $r_{\text{end}_{(x,y_1,y_2)}}$ uniformly and independently at random from $\mathbb{Z}_p$. Compute the elements of the key as follows.

$$K_{\text{start},1} = D_{0,0,0} + r_{\text{start}} H_{\text{start}}, \quad K_{\text{start},2} = r_{\text{start}} P,$$

for all $t \in \mathcal{T}$ with $t = (q_x, q_{x'}, \sigma, Z_{y_1}, Z_{y_2} Z_{y_1'}, Z_{y_2'})$ and $\sigma \in \Sigma \cup \{\lambda\}$,

$$K_{t,1} = -D_{x,y_1,y_2} + r_t P_1, \quad K_{t,2} = r_t P, \quad K_{t,3} = D_{x',y_1',y_2'} + r_t H_\sigma,$$

for all $q_x \in F$ and for all $Z_{y_1}, Z_{y_2} \in \Gamma$,

$$K_{\text{end}_{(x,y_1,y_2)},1} = -\alpha P + D_{x,y_1,y_2} + r_{\text{end}_{(x,y_1,y_2)}} H_{\text{end}}, \quad K_{\text{end}_{(x,y_1,y_2)},2} = r_{\text{end}_{(x,y_1,y_2)}} P.$$

The secret key for automaton $\mathcal{M}$ is given by $\mathcal{SK}_\mathcal{M} = (K_{\text{start},1}, K_{\text{start},2}, \{K_{t,1}, K_{t,2}, K_{t,3}\}_{t\in\mathcal{T}}, \{K_{\text{end}_{(x,y_1,y_2)},1}, K_{\text{end}_{(x,y_1,y_2)},2}\}_{q_x\in F, Z_{y_1}, Z_{y_2}\in\Gamma})$.

$\mathsf{Decrypt}(\mathcal{C}, \mathcal{SK}_\mathcal{M})$: Suppose that $\mathsf{Accept}(\mathcal{M}, w) = 1$. Then there exists a sequence of transitions $t_1, t_2, \ldots, t_k$ with $t_i = (q_{x_{i-1}}, q_{x_i}, \sigma_i, Z_{y_{1,i-1}}, Z_{y_{2,i-1}}, Z_{y_{1,i}}, Z_{y_{2,i}})$ where $x_0 = 0$, $y_{1,0} = y_{2,0} = 0$, $x_1, \ldots, x_k \in \mathbb{Z}_{|Q|}$, $y_{1,i}, y_{2,i} \in \mathbb{Z}_{|\Gamma|}$ and $(\sigma_i = \lambda) \wedge (\sigma_i = w_j)$ for some j. Decryption consists of several stages of computation. Each stage computes an intermediate value $A_{i,j} = e(P, D_{x_i,y_{1,i},y_{2,i}})^{s_j}$ marking the end of i-th transition and j many input symobls being read. At the beginning $A_{0,0}$ is computed as

$$\begin{aligned}A_{0,0} &= e(C_{\text{start},1}, K_{\text{start},1}) e(C_{\text{start},2}, K_{\text{start},2})^{-1}\\ &= e(s_0 P, D_{0,0,0} + r_{\text{start}} H_{\text{start}}) e(s_0 H_{\text{start}}, r_{\text{start}} P)^{-1}\\ &= e(P, D_{0,0,0})^{s_0} e(P, H_{\text{start}})^{s_0 r_{\text{start}}} e(H_{\text{start}}, P)^{-s_0 r_{\text{start}}}\\ &= e(P, D_{0,0,0})^{s_0}.\end{aligned}$$

$A_{i,j}$'s are computed at the end of each transition. For $i = 1, \ldots, k$, depending on whether t_i is a λ-transition or not, two cases are considered.

Case 1: $\sigma_i = w_j$ and $j-1$ symbols have been read.

In this case, $A_{i,j}$ is given by

$$\begin{aligned}
A_{i,j} &= A_{i-1,j-1} \cdot e(C_{j-1,1}, K_{t_i,1}) e(C_{j,2}, K_{t_i,2})^{-1} e(C_{j,1}, K_{t_i,3}) \\
&= e(P, D_{x_{i-1},y_{1,i-1},y_{2,i-1}})^{s_{j-1}} e(s_{j-1}P, -D_{x_{i-1},y_{1,i-1},y_{2,i-1}} + r_{t_i}P_1) \\
&\quad \cdot e(s_j H_{w_j} + s_{j-1}P_1, r_{t_i}P)^{-1} e(s_j P, D_{x_i,y_{1,i},y_{2,i}} + r_{t_i}H_{\sigma_i}) \\
&= e(P, D_{x_{i-1},y_{1,i-1},y_{2,i-1}})^{s_{j-1}} e(P, D_{x_{i-1},y_{1,i-1},y_{2,i-1}})^{-s_{j-1}} e(P, P_1)^{s_{j-1}r_{t_i}} \\
&\quad \cdot e(H_{w_j}, P)^{-s_j r_{t_i}} e(P_1, P)^{-s_{j-1}r_{t_i}} e(P, D_{x_i,y_{1,i},y_{2,i}})^{s_j} e(P, H_{\sigma_i})^{s_j r_{t_i}} \\
&= e(H_{w_j}, P)^{-s_j r_{t_i}} e(P, D_{x_i,y_{1,i},y_{2,i}})^{s_j} e(P, H_{w_j})^{s_j r_{t_i}} \\
&= e(P, D_{x_i,y_{1,i},y_{2,i}})^{s_j}.
\end{aligned}$$

Case 2: $\sigma_i = \lambda$ and j symbols have been read.

Compute $A_{i,j}$ as

$$\begin{aligned}
A_{i,j} &= A_{i-1,j} \cdot e(C_{j,1}, K_{t_i,1}) e(C_{j+1,3}, K_{t_i,2})^{-1} e(C_{j,1}, K_{t_i,3}) \\
&= e(P, D_{x_{i-1},y_{1,i-1},y_{2,i-1}})^{s_j} e(s_j P, -D_{x_{i-1},y_{1,i-1},y_{2,i-1}} + r_{t_i}P_1) \\
&\quad \cdot e(s_j H_\lambda + s_j P_1, r_{t_i}P)^{-1} e(s_j P, D_{x_i,y_{1,i},y_{2,i}} + r_{t_i}H_\lambda) \\
&= e(P, D_{x_{i-1},y_{1,i-1},y_{2,i-1}})^{s_j} e(P, D_{x_{i-1},y_{1,i-1},y_{2,i-1}})^{-s_j} e(P, P_1)^{s_j r_{t_i}} \\
&\quad \cdot e(H_\lambda, P)^{-s_j r_{t_i}} e(P_1, P)^{-s_j r_{t_i}} e(P, D_{x_i,y_{1,i},y_{2,i}})^{s_j} e(P, H_\lambda)^{s_j r_{t_i}} \\
&= e(P, D_{x_i,y_{1,i},y_{2,i}})^{s_j}.
\end{aligned}$$

At the end, $A_{k,\ell} = e(P, D_{x_k,y_{1,k},y_{2,k}})^{s_\ell}$ is used to unmask the message as shown below.

$$\begin{aligned}
&C_m \cdot e(C_{\mathrm{end},1}, K_{\mathrm{end}_{(x_k,y_k)},1}) \cdot e(C_{\mathrm{end},2}, K_{\mathrm{end}_{(x_k,y_k)},2})^{-1} \cdot A_{k,\ell}^{-1} \\
&= m \cdot e(P,P)^{\alpha s_l} \cdot e(s_\ell P, -\alpha P + D_{x_k,y_{1,k},y_{2,k}} + r_{\mathrm{end}_{x_k,y_{1,k},y_{2,k}}} H_{\mathrm{end}}) \\
&\quad \cdot e(s_\ell H_{\mathrm{end}}, r_{\mathrm{end}_{x_k,y_{1,k},y_{2,k}}} P)^{-1} \cdot e(P, D_{x_k,y_{1,k},y_{2,k}})^{-s_\ell} \\
&= m \cdot e(P,P)^{\alpha s_l} \cdot e(P,P)^{-\alpha s_\ell} e(P, D_{x_k,y_{1,k},y_{2,k}})^{s_\ell} e(P, H_{\mathrm{end}})^{s_\ell r_{\mathrm{end}_{x_k,y_{1,k},y_{2,k}}}} \\
&\quad \cdot e(H_{\mathrm{end}}, P)^{-s_\ell r_{\mathrm{end}_{x_k,y_{1,k},y_{2,k}}}} \cdot e(P, D_{x_k,y_{1,k},y_{2,k}})^{-s_\ell} \\
&= m
\end{aligned}$$

Suppose w is accepted by $\mathcal{M}$. By construction, if the acceptance occurs before $\mathcal{M}$ completely reads w, then the decryption will not work. This is because the randomiser in the mask $e(P,P)^{\alpha s_\ell}$ is s_ℓ which is provided with the component of the ciphertext corresponding to the last symbol w_ℓ of w. This is the reason why we require $\mathcal{M}$ to completely read the input string before halting.

Discussion. We outline the main differences between our construction and that of Waters [Wat12].

1. In a pushdown automata, we need to keep track of both states and stack contents. This is done by choosing group elements D_{x,y_1,y_2} corresponding to all combinations of state and top-of-stacks (x, y_1, y_2) where x represents the state and y_1, y_2 represent tops of the two stacks. Note that it is not required to keep track of all the stack contents. The transitions are determined depending only on what is at the top.

2. A new element H_λ is introduced in the public parameters that corresponds to the empty string. Ciphertext for a string w contains ℓ extra elements $C_{i,3} = s_{i-1}H_\lambda + s_{i-1}P_1$ for $0 \leq i \leq \ell - 1$. The key components corresponding to λ-transitions will have the same structure as that for any other input symbol $\sigma \in \Sigma$ except that H_λ is used in constructing them (instead of H_σ). Consider decryption of a ciphertext $\mathcal{C}$ encrypted to $w = w_1 \cdots w_\ell$ by a secret key $\mathcal{SK}_\mathcal{M}$. With the presence of λ-transitions, the number of transitions that $\mathcal{M}$ makes during decryption may be greater than ℓ. Suppose that there are k transitions. At the end of the k-th transition, we must have the intermediate value $e(P, D_{x_k, y_{1,k}, y_{2,k}})^{s_j}$ if j symbols have been read. Now look at the $(k+1)$-st transition. If it is a λ-transition, then the input pointer is not advanced and hence the exponent must remain s_j even though the state changes. Otherwise it is not possible to cancel it out when the next symbol is read since the group element corresponding to the next symbol $H_{w_{j+1}}$ is randomised by s_{j+1}. This is why we provide the third component $C_{i,3}$ for every i and it consists of H_λ randomised with s_{i-1} so that the exponent remains unchanged.

Note 4.1. As discussed earlier, to cover the class recursive languages it is sufficient to be able to handle 2DPDAs. We note though that the above construction easily extends to k-stack DPDAs for any $k \geq 1$. This will not require any change in the public parameters or the ciphertext. The only change will be in the key generation where the D's will have to be subscripted by a $(k+1)$-length vector. One possible reason for preferring a DPDA with more stacks could be that the time taken to halt (and hence by the decryption algorithm) may be faster.

4.2 ℓ-XBDHE Assumption

Provided here is a description of the eXpanded ℓ-Bilinear Diffie-Hellman Exponent (ℓ-XBDHE) assumption used in [Wat12]. Let $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, e, P)$ be a symmetric pairing and $a, b, c_0, \ldots, c_{\ell+1}, s, d$ be elements chosen uniformly and independently at random from $\mathbb{Z}_p$. Define a distribution $\mathcal{D}$ as consisting of the following elements.

$$P,\ aP,\ bP,\ (ab/d)P,\ (b/d)P$$

$$\forall i \in [0, 2\ell+1] \setminus \{\ell+1\}, \forall j \in [0, \ell+1], \quad a^i sP,\ (a^i bs/c_j)P$$

$$\forall i \in [0, \ell+1], \quad (a^i b/c_i)P,\ c_iP,\ a^i dP,\ (abc_i/d)P,\ (bc_i/d)P$$

$$\forall i \in [0, 2\ell+1], \forall j \in [0, \ell+1], \quad (a^i bd/c_j)P$$

$$\forall i, j \in [0, \ell+1], i \neq j, \quad (a^i bc_j/c_i)P.$$

For an algorithm $\mathscr{A}$ that returns a bit, define its advantage in solving the ℓ-XBDHE problem as

$$\mathsf{Adv}_{\mathcal{G}}^{\ell\text{-XBDHE}}(\mathscr{A}) = \left| \Pr[\mathscr{A}(\mathcal{D}, e(P,P)^{a^{i+1}bs}) = 1] - \Pr[\mathscr{A}(\mathcal{D}, X_T) = 1] \right|,$$

where $X_T \in_{\mathrm{U}} \mathbb{G}_T$. The (t, ε)-ℓ-XBDHE assumption is said to hold if for every algorithm $\mathscr{A}$ running in time at most t, $\mathsf{Adv}_{\mathcal{G}}^{\ell\text{-XBDHE}}(\mathscr{A}) \leq \varepsilon$.

4.3 Security in the Selective Model

We prove security of the above construction in the selective model under the ℓ-XBDHE assumption. The structure of the proof is similar to that of [Wat12]. Important changes are mainly in the key extraction phase where the simulator has to handle stack contents as well as λ-transitions of the queried automata.

Theorem 4.1. *If the* (t, ε)-ℓ-XBDHE *assumption holds then the FE scheme of Section 4.1 is* (ε', t', ν)-IND-STR-CPA *secure, where* $\varepsilon' \leq \varepsilon$ *and* $t' = t - O(\nu\ell^*|\Sigma|\rho)$ *(*ρ *denotes the time required for one scalar multiplication in* $\mathbb{G}$*).*

Proof. Let $\text{Game}_{\text{real}}$ denote the actual selective security game ind-s-cpa as defined in Section 3.3. Define $\text{Game}_{\text{random}}$ to be similar to $\text{Game}_{\text{real}}$ except that the challenge ciphertext returned to the adversary is an encryption of a random message.

Initialise: The adversary commits to an input alphabet Σ and a challenge string $w^* = w_1^* \cdots w_{\ell^*}^*$.

Setup: The simulator $\mathscr{B}$ obtains an instance of the ℓ^*-XBDHE assumption consisting of $\mathcal{D}, X_T$ where $\mathcal{D}$ is as defined in Section 4.2 and X_T is an element of $\mathbb{G}_T$ whose distribution $\mathscr{B}$ is supposed to guess. Let $w_0^* = \$$ and $w_{\ell^*+1}^* = \$$. $\mathscr{B}$ chooses $v_1, v_{\text{start}}, v_{\text{end}}, v_\lambda, \{v_\sigma\}_{\sigma\in\Sigma} \xleftarrow{\text{U}} \mathbb{Z}_p$ and sets the parameters as follows:

$$e(P,P)^\alpha = e(aP, bP), \; P_1 = v_1 P + (ab/d)P, \; H_\lambda = v_\lambda P - (ab/d)P$$

$$H_{\text{start}} = v_{\text{start}}P - \sum_{j=1}^{\ell^*}(a^j b/c_j)P, \; H_{\text{end}} = v_{\text{end}}P - \sum_{j=2}^{\ell^*+1}(a^j b/c_j)P,$$

$$\text{for all } \sigma \in \Sigma, \; H_\sigma = v_\sigma P - (b/d)P - \sum_{\substack{j\in[0,\ell^*+1] \\ w_j^*\neq\sigma}} (a^{\ell^*+1-j} b/c_{\ell^*+1-j})P,$$

implicitly setting $\alpha = ab$. $\mathscr{B}$ provides the public parameters to $\mathscr{A}$. Note that all the elements required to construct the parameters are available in the problem instance. Also, the H_σ's are well-defined and properly distributed. Each H_σ will certainly contain the terms $(ab/c_0)P$ and $(a^{\ell^*+1}b/c_{\ell^*+1})P$. The choice of v_λ and v_1 ensures that elements H_λ and P_1 are uniformly and independently distributed.

Key Generation Phases 1 and 2: The adversary makes several (at most ν) key extraction queries. Let $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, Z_0, Z_0, F, \delta)$ be an automaton for which the adversary requests a key. $\mathscr{B}$ creates a key for $\mathcal{M}$ only if $\mathsf{Accept}(\mathcal{M}, w^*) = 0$. Now we describe how the key is constructed. Define index sets S_{x,y_1,y_2} for $(x, y_1, y_2) \in \mathbb{Z}_{|Q|} \times \mathbb{Z}_{|\Gamma|} \times \mathbb{Z}_{|\Gamma|}$ as

$$S_{x,y_1,y_2} = \{i \in [0, \ell^*] : \mathsf{Accept}(\mathcal{M}_{x,y_1,y_2}, w^{*(i)}) = 1\}.$$

Here $w^{*(i)}$ denotes the suffix of length i of w^* (i.e., the string containing last i symbols of w^*). The automaton $\mathcal{M}_{x,y_1,y_2}$ is identical to $\mathcal{M}$ except that the start state is q_x and the bottom-of-stack symbols are Z_{y_1} and Z_{y_2} for the two stacks respectively. Set $D_{x,y_1,y_2} = \sum_{i\in S_{x,y_1,y_2}} a^{i+1}bP$ for all pairs $(x, y_1, y_2) \in \mathbb{Z}_{|Q|} \times \mathbb{Z}_{|\Gamma|} \times \mathbb{Z}_{|\Gamma|}$. Observe that none of the D_{x,y_1,y_2}'s are known. The other randomisers must be properly chosen from the instance so as to cancel out these elements.

The simulator then computes the different components of the key. It implicitly sets $r_{\text{start}} = \sum_{i\in S_{0,0,0}} c_{i+1}$ and computes

$$K_{\text{start},2} = \sum_{i\in S_{0,0,0}} c_{i+1}P, \quad K_{\text{start},1} = v_{\text{start}}K_{\text{start},2} - \sum_{\substack{j\in[1,\ell^*] \\ i\in S_{0,0,0} \\ j\neq i+1}} (a^j b c_{i+1}/c_j)P.$$

The terms of the summation in $K_{\text{start},1}$ with $j = i + 1$ i.e., $c_j = c_{i+1}$ get canceled with the terms in $D_{0,0,0}$

as shown in the following computation.

$$\begin{aligned}
K_{\text{start},1} &= D_{0,0,0} + r_{\text{start}} H_{\text{start}} \\
&= \sum_{i \in S_{0,0,0}} a^{i+1} bP + \left(\sum_{i \in S_{0,0,0}} c_{i+1} \right) \left(v_{\text{start}} P - \sum_{j=1}^{\ell^*} (a^j b / c_j) P \right) \\
&= \sum_{i \in S_{0,0,0}} a^{i+1} bP + v_{\text{start}} \sum_{i \in S_{0,0,0}} c_{i+1} P - \sum_{\substack{j \in [1,\ell^*] \\ i \in S_{0,0,0}}} (a^j b c_{i+1} / c_j) P \\
&= \sum_{i \in S_{0,0,0}} a^{i+1} bP + v_{\text{start}} \sum_{i \in S_{0,0,0}} c_{i+1} P - \sum_{\substack{i \in [0,\ell^*-1] \\ i \in S_{0,0,0}}} a^{i+1} bP - \sum_{\substack{j \in [1,\ell^*] \\ i \in S_{0,0,0} \\ j \neq i+1}} (a^j b c_{i+1} / c_j) P \\
&= \sum_{i \in S_{0,0,0}} a^{i+1} bP + v_{\text{start}} \sum_{i \in S_{0,0,0}} c_{i+1} P - \sum_{i \in S_{0,0,0}} a^{i+1} bP - \sum_{\substack{j \in [1,\ell^*] \\ i \in S_{0,0,0} \\ j \neq i+1}} (a^j b c_{i+1} / c_j) P \\
&= v_{\text{start}} K_{\text{start},2} - \sum_{\substack{j \in [1,\ell^*] \\ i \in S_{0,0,0} \\ j \neq i+1}} (a^j b c_{i+1} / c_j) P
\end{aligned}$$

We used the fact that the condition $(i \in [0, \ell^* - 1]) \wedge (i \in S_{0,0,0})$ is equivalent to $i \in S_{0,0,0}$. This is because $S_{0,0,0}$ does not contain ℓ^*; otherwise, the automaton $\mathcal{M}_{0,0,0}\,(= \mathcal{M})$ would accept $w^{*(\ell^*)} = w^*$ and the game disallows such queries.

Next, $\mathscr{B}$ implicitly sets $r_{\text{end}_{(x,y_1,y_2)}} = \sum_{\substack{i \in S_{x,y_1,y_2} \\ i \neq 0}} c_{i+1}$ for all $q_x \in F$ and $Z_{y_1}, Z_{y_2} \in \Gamma$ and computes

$$K_{\text{end}_{(x,y_1,y_2)},2} = \sum_{\substack{i \in S_{x,y_1,y_2} \\ i \neq 0}} c_{i+1} P, \quad K_{\text{end}_{(x,y_1,y_2)},1} = v_{\text{end}} K_{\text{end}_{(x,y_1,y_2)},2} - \sum_{\substack{j \in [2,\ell^*+1] \\ i \in S_{x,y_1,y_2} \\ j \neq i+1 \\ i \neq 0}} (a^j b c_{i+1} / c_j) P.$$

Again, the terms in the summation above with $j = i + 1$ and $i \neq 0$ get canceled with those in D_{x,y_1,y_2}. The only term in D_{x,y_1,y_2} that remains is $a^1 bP$. However, this gets canceled with $-\alpha P = -abP$. Element D_{x,y_1,y_2} does contain abP since q_x is a final state and consequently, the suffix of length 0 i.e., the empty string is accepted by $\mathcal{M}_{x,y_1,y_2}$. Detailed calculations are provided below.

$$\begin{aligned}
K_{\mathrm{end}_{(x,y_1,y_2)},1} &= -\alpha P + D_{x,y_1,y_2} + r_{\mathrm{end}_{(x,y_1,y_2)}} H_{\mathrm{end}} \\
&= -abP + \sum_{i \in S_{x,y_1,y_2}} a^{i+1}bP + \left(\sum_{\substack{i \in S_{x,y_1,y_2} \\ i \neq 0}} c_{i+1} \right) \left(v_{\mathrm{end}}P - \sum_{j=2}^{\ell^*+1} (a^j b / c_j) P \right) \\
&= -abP + abP + \sum_{\substack{i \in S_{x,y_1,y_2} \\ i \neq 0}} a^{i+1}bP + v_{\mathrm{end}} \sum_{\substack{i \in S_{x,y_1,y_2} \\ i \neq 0}} c_{i+1}P - \sum_{\substack{j \in [2,\ell^*+1] \\ i \in S_{x,y_1,y_2} \\ i \neq 0}} (a^j b c_{i+1}/c_j)P \\
&= \sum_{\substack{i \in S_{x,y_1,y_2} \\ i \neq 0}} a^{i+1}bP + v_{\mathrm{end}} \sum_{\substack{i \in S_{x,y_1,y_2} \\ i \neq 0}} c_{i+1}P - \sum_{\substack{j \in [2,\ell^*+1] \\ i \in S_{x,y_1,y_2} \\ j = i+1 \\ i \neq 0}} (a^j b c_{i+1}/c_j)P - \sum_{\substack{j \in [2,\ell^*+1] \\ i \in S_{x,y_1,y_2} \\ j \neq i+1 \\ i \neq 0}} (a^j b c_{i+1}/c_j)P \\
&= \sum_{\substack{i \in S_{x,y_1,y_2} \\ i \neq 0}} a^{i+1}bP + v_{\mathrm{end}} \sum_{\substack{i \in S_{x,y_1,y_2} \\ i \neq 0}} c_{i+1}P - \sum_{\substack{i \in S_{x,y_1,y_2} \\ i \neq 0}} a^{i+1}bP - \sum_{\substack{j \in [2,\ell^*+1] \\ i \in S_{x,y_1,y_2} \\ j \neq i+1 \\ i \neq 0}} (a^j b c_{i+1}/c_j)P \\
&= v_{\mathrm{end}} K_{\mathrm{end}_{(x,y_1,y_2)},2} - \sum_{\substack{j \in [2,\ell^*+1] \\ i \in S_{x,y_1,y_2} \\ j \neq i+1 \\ i \neq 0}} (a^j b c_{i+1}/c_j)P
\end{aligned}$$

The next step is to construct the elements $K_{t,1}, K_{t,2}, K_{t,3}$ for each $t = (x, x', \sigma, y_1, y_2, y_1', y_2') \in \mathcal{T}$. $K_{t,1}$ and $K_{t,3}$ consist of the elements $D_{x,y_1,y_2} = \left(\sum_{i \in S_{x,y_1,y_2}} a^{i+1}b\right)$ and $D_{x',y_1',y_2'} = \left(\sum_{i \in S_{x',y_1',y_2'}} a^{i+1}b\right)$ respectively. These elements cannot be computed since $a^{i+1}b$ (for $i \in [0,\ell^*]$) are not available from the problem instance. The trick is to compute $K_{t,1}, K_{t,2}, K_{t,3}$ even without knowing the D's. The only thing that we can choose is the randomiser r_t and an appropriate choice makes it possible to compute $K_{t,\cdot}$'s. For each $i \in [0,\ell^*]$, we choose a sub-randomiser $r_{t,i}$ so that $a^{i+1}b$ and a^ib (or $a^{i+1}b$ if t is a λ-transition), if present in D_{x,y_1,y_2} and $D_{x',y_1',y_2'}$ (respectively), are cancelled out. Also, it must be possible to construct $K_{t,2} = r_tP = \sum_{i \in [0,\ell^*]} r_{t,i}P$ using elements provided in the instance. The sum of these sub-randomisers gives us r_t (i.e, $r_t = \sum_{i \in [0,\ell^*]} r_{t,i}$).

Let $K_{t,\cdot,i}$ denote the component of $K_{t,\cdot}$ corresponding to $r_{t,i}$. The elements $K_{t,1}, K_{t,3}$ are now given by

$$K_{t,1} = \sum_{i=0}^{\ell^*} K_{t,1,i}, \quad K_{t,3} = \sum_{i=0}^{\ell^*} K_{t,3,i}.$$

We now move on to choosing the sub-randomisers $r_{t,i}$ depending on whether t is a λ-transition or not. If t is not a λ-transition, i.e., if $\sigma \neq \lambda$, then there four possible cases.

Case 1: $(i \notin S_{x,y_1,y_2}) \wedge (i - 1 \notin S_{x',y_1',y_2'})$

In this case, there are no terms to cancel out. We set $r_{t,i} = 0$ and hence $K_{t,1,i} = K_{t,2,i} = K_{t,3,i} = 0$.

Case 2: $(i \in S_{x,y_1,y_2}) \wedge (i-1 \in S_{x',y_1',y_2'})$

The terms $a^{i+1}bP$ and a^ibP are present in D_{x,y_1,y_2} and $D_{x',y_1',y_2'}$ (resp.) and in order to eliminate them, we set $r_{t,i} = a^id$ and $K_{t,2,i} = a^idP$. The other components are set as follows.

$$K_{t,1,i} = v_1 K_{t,2,i}, \quad K_{t,3,i} = v_\sigma K_{t,2,i} - \sum_{\substack{j \in [0,\ell^*+1] \\ w_j^* \neq \sigma}} (a^{\ell^*+1-j+i} bd/c_{\ell^*+1-j})P.$$

Recall that D_{x,y_1,y_2} (resp. $a^{i+1}bP$) appears with a negative sign in $K_{t,1}$ (resp. $K_{t,1,i}$). The calculations given below explain how the terms $-a^{i+1}bP$ and a^ibP are canceled out in $K_{t,1,i}$ and $K_{t,2,i}$.

$$\begin{aligned}
K_{t,1,i} &= -a^{i+1}bP + r_{t,i}P_1 \\
&= -a^{i+1}bP + a^id\,(v_1P + (ab/d)P) \\
&= -a^{i+1}bP + v_1(a^idP) + a^{i+1}bP \\
&= v_1K_{t,2,i} \\
K_{t,3,i} &= a^ibP + r_{t,i}(H_\sigma) \\
&= a^ibP + a^id\left(v_\sigma P - (b/d)P - \sum_{\substack{j \in [0,\ell^*+1] \\ w_j^* \neq \sigma}} (a^{\ell^*+1-j} b/c_{\ell^*+1-j})P \right) \\
&= a^ibP + v_\sigma(a^idP) - a^ibP - \sum_{\substack{j \in [0,\ell^*+1] \\ w_j^* \neq \sigma}} (a^{\ell^*+1-j+i} bd/c_{\ell^*+1-j})P \\
&= v_\sigma K_{t,2,i} - \sum_{\substack{j \in [0,\ell^*+1] \\ w_j^* \neq \sigma}} (a^{\ell^*+1-j+i} bd/c_{\ell^*+1-j})P
\end{aligned}$$

Case 3: $(i \notin S_{x,y_1,y_2}) \wedge (i-1 \in S_{x',y_1',y_2'})$

Suppose that w_{ℓ^*+1-i} (the first symbol of $w^{*(i)}$) is equal to σ. Then $\mathcal{M}_{x,y_1,y_2}$, on reading input symbol σ, must arrive at state $q_{x'}$ with the top of stacks being $Z_{y_1'}$ and $Z_{y_2'}$. By definition, $i-1 \in S_{x',y_1',y_2'}$ indicates that $\mathcal{M}_{x',y_1',y_2'}$ accepts $w^{*(i-1)}$. This would in turn imply that $\mathcal{M}_{x,y_1,y_2}$ accepts the string $w^{*(i)}$, which clearly violates the condition $i \notin S_{x,y_1,y_2}$. Therefore, $w^*_{\ell^*+1-i} \neq \sigma$.

Set $r_{t,i} = c_i$, $K_{t,2,i} = c_iP$ and the remaining two components as

$$K_{t,1,i} = v_1K_{t,2,i} + (abc_i/d)P, \quad K_{t,3,i} = v_\sigma K_{t,2,i} - (bc_i/d)P - \sum_{\substack{j \in [0,\ell^*+1] \\ w_j^* \neq \sigma \\ j \neq \ell^*+1-i}} (a^{\ell^*+1-j} bc_i/c_{\ell^*+1-j})P.$$

The well-formedness of these components can be verified via the following calculations.

$$\begin{aligned}
K_{t,1,i} &= r_{t,i}P_1 = c_iP_1 = c_i(v_1P + (ab/d)P) = v_1K_{t,2,i} + (abc_i/d)P \\
K_{t,3,i} &= a^ibP + r_{t,i}H_\sigma \\
&= a^ibP + c_iH_\sigma \\
&= a^ibP + c_i\left(v_\sigma P - (b/d)P - \sum_{\substack{j\in[0,\ell^*+1]\\ w_j^*\neq\sigma}} (a^{\ell^*+1-j}b/c_{\ell^*+1-j})P\right) \\
&= a^ibP + v_\sigma c_iP - (bc_i/d)P - a^ibP - \sum_{\substack{j\in[0,\ell^*+1]\\ w_j^*\neq\sigma\\ j\neq\ell^*+1-i}} (a^{\ell^*+1-j}bc_i/c_{\ell^*+1-j})P \\
&= v_\sigma K_{t,2,i} - (bc_i/d)P - \sum_{\substack{j\in[0,\ell^*+1]\\ w_j^*\neq\sigma\\ j\neq\ell^*+1-i}} (a^{\ell^*+1-j}bc_i/c_{\ell^*+1-j})P
\end{aligned}$$

Case 4: $(i \in S_{x,y_1,y_2}) \wedge (i-1 \notin S_{x',y_1',y_2'})$

Due to a similar reason as explained in **Case 3**, it must hold that $w^*_{\ell^*+1-i} \neq \sigma$. The randomiser $r_{t,i}$ is set to $a^id - c_i$ and components of the key are computed as follows.

$$K_{t,2,i} = a^idP - c_iP, \quad K_{t,1,i} = v_1K_{t,2,i} - (abc_i/d)P,$$

$$K_{t,3,i} = v_\sigma K_{t,2,i} + (bc_i/d)P - \sum_{\substack{j\in[0,\ell^*+1]\\ w_j^*\neq\sigma}} (a^{\ell^*+1-j+i}bd/c_{\ell^*+1-j})P + \sum_{\substack{j\in[0,\ell^*+1]\\ w_j^*\neq\sigma\\ j\neq\ell^*+1-i}} (a^{\ell^*+1-j}bc_i/c_{\ell^*+1-j})P.$$

This case can be seen as a combination of cases 2 and 3. We set $r_{t,i} = a^idP$ which cancels out with $a^{i+1}bP$ present in D_{x,y_1,y_2}. But this results in an extra term $-a^ibP$ in $K_{t,3,i}$ that cannot be canceled out by $D_{x',y_1',y_2'}$ since it does not contain a^ibP. Therefore, the $-c_i$ is added to $r_{t,i}$ in order to eliminate the extra term. Detailed calculations are given below.

$$\begin{aligned}
K_{t,1,i} &= -a^{i+1}bP + r_{t,i}P_1 \\
&= -a^{i+1}bP + (a^id - c_i)P_1 \\
&= -a^{i+1}bP + (a^id - c_i)\left(v_1P + (ab/d)P\right) \\
&= -a^{i+1}bP + v_1(a^id - c_i) - (abc_i/d)P + a^{i+1}bP \\
&= v_1K_{t,2,i} - (abc_i/d)P \\
K_{t,3,i} &= r_{t,i}H_\sigma \\
&= (a^id - c_i)H_\sigma \\
&= (a^id - c_i)\left(v_\sigma P - (b/d)P - \sum_{\substack{j\in[0,\ell^*+1]\\ w_j^*\neq\sigma}} (a^{\ell^*+1-j}b/c_{\ell^*+1-j})P\right) \\
&= (a^id - c_i)v_\sigma P - a^ibP + (bc_i/d)P - \sum_{\substack{j\in[0,\ell^*+1]\\ w_j^*\neq\sigma}} (a^{\ell^*+1-j+i}bd/c_{\ell^*+1-j})P \\
&\quad + \sum_{\substack{j\in[0,\ell^*+1]\\ w_j^*\neq\sigma}} (a^{\ell^*+1-j}bc_i/c_{\ell^*+1-j})P \\
&= v_\sigma K_{t,2,i} - a^ibP + (bc_i/d)P - \sum_{\substack{j\in[0,\ell^*+1]\\ w_j^*\neq\sigma}} (a^{\ell^*+1-j+i}bd/c_{\ell^*+1-j})P \\
&\quad + a^ibP + \sum_{\substack{j\in[0,\ell^*+1]\\ w_j^*\neq\sigma\\ j\neq\ell^*+1-i}} (a^{\ell^*+1-j}bc_i/c_{\ell^*+1-j})P
\end{aligned}$$

We next consider the the possiblity of t being a λ-transition i.e., $\sigma = \lambda$. Only one of the following two cases can occur.

Case 1: $(i \notin S_{x,y_1,y_2}) \wedge (i \notin S_{x',y_1',y_2'})$

Neither D_{x,y_1,y_2} nor $D_{x',y_1',y_2'}$ would contain the term $a^{i+1}bP$ and hence we set $r_{t,i} = 0$ and $K_{t,1,i} = K_{t,2,i} = K_{t,3,i} = 0$.

Case 2: $(i \in S_{x,y_1,y_2}) \wedge (i \in S_{x',y_1',y_2'})$

Both D_{x,y_1,y_2} and $D_{x',y_1',y_2'}$ contain the term $a^{i+1}bP$ and to cancel it out we set $K_{t,2,i} = a^idP$, $K_{t,1,i} = v_1K_{t,2,i}$ and $K_{t,3,i} = v_\lambda K_{t,2,i}$ with $r_{t,i} = a^id$. Note that the components are well-formed as shown below.

$$K_{t,1,i} = -a^{i+1}bP + r_{t,i}P_1 = -a^{i+1}bP + a^id(v_1P + (ab/d)P) = v_1K_{t,2,i}$$

$$K_{t,2,i} = a^{i+1}bP + r_{t,i}H_\lambda = a^{i+1}bP + a^id(v_\lambda P - (ab/d)P) = v_\lambda K_{t,2,i}$$

Let $\mathcal{SK}_\mathcal{M} = (K_{\text{start},1}, K_{\text{start},2}, \{K_{t,1}, K_{t,2}, K_{t,3}\}_{t\in\mathcal{T}}, \{K_{\text{end}_{(x,y_1,y_2)},1}, K_{\text{end}_{(x,y_1,y_2)},2}\}_{q_x\in F, Z_{y_1}, Z_{y_2}\in\Gamma})$. It is clear that the components of $\mathcal{SK}_\mathcal{M}$ are not properly distributed. $\mathscr{B}$ must re-randomise them suitably before providing them to the adversary. We use an algorithm $\mathsf{ReRandK}$ described in Section 4.4 that re-randomises keys. The key returned to $\mathscr{A}$ is the output of $\mathsf{ReRandK}(\mathcal{SK}_{\mathcal{M}})$.

Challenge Phase: $\mathscr{A}$ provides two messages m_0, m_1 to $\mathscr{B}$. The simulator chooses $\beta \xleftarrow{\mathrm{U}} \{0,1\}$ and encrypts m_β under the challenge string w^*. The randomiser s_i is implicitly set to $a^i s$ for each $i = 0, \ldots, \ell^*$.

$$C_m = m_\beta \cdot X_T,$$

$$C_{\text{start},1} = sP, \quad C_{\text{start},2} = v_{\text{start}}(sP) - \sum_{j=1}^{\ell^*}(a^j bs/c_j)P,$$

For $i = 1, \ldots, \ell^*$,

$$C_{i,1} = a^i sP, \quad C_{i,3} = (v_1 + v_\lambda)(a^{i-1}sP),$$
$$C_{i,2} = v_{w_i^*}(a^i sP) + v_1(a^{i-1}sP) - \sum_{\substack{j\in[0,\ell^*+1] \\ w_j^* \neq w_i^*}} (a^{\ell^*+1-j+i}bs/c_{\ell^*+1-j})P,$$

$$C_{\text{end},1} = a^{\ell^*}sP, \quad C_{\text{end},2} = v_{\text{end}}(a^{\ell^*}sP) - \sum_{2}^{\ell^*+1}(a^{j+\ell^*}bs/c_j)P.$$

Let $\mathcal{C}$ be the ciphertext containing the above elements. The following comptuation shows the well-formedness of $C_{i,2}$ and $C_{i,3}$.

$$\begin{aligned}
C_{i,3} &= s_{i-1}H_\lambda + s_{i-1}P_1 \\
&= a^{i-1}s\,(v_\lambda P - (ab/d)P) + a^{i-1}s\,(v_1 P + (ab/d)P) \\
&= (v_1 + v_\lambda)(a^{i-1}sP) \\
C_{i,2} &= s_i H_{w_i} + s_{i-1}P_1 \\
&= a^i sP \left(v_{w_i^*}P - (b/d)P - \sum_{\substack{j\in[0,\ell^*+1] \\ w_j^* \neq w_i^*}} (a^{\ell^*+1-j}b/c_{\ell^*+1-j})P \right) + a^{i-1}sP(v_1 P + (ab/d)P) \\
&= v_{w_i^*}(a^i sP) - (a^i sb/d)P - \sum_{\substack{j\in[0,\ell^*+1] \\ w_j^* \neq w_i^*}} (a^{\ell^*+1-j+i}bs/c_{\ell^*+1-j})P + v_1(a^{i-1}sP) + (a^i sb/d)P \\
&= v_{w_i^*}(a^i sP) + v_1(a^{i-1}sP) - \sum_{\substack{j\in[0,\ell^*+1] \\ w_j^* \neq w_i^*}} (a^{\ell^*+1-j+i}bs/c_{\ell^*+1-j})P
\end{aligned}$$

Since $w_j^* \neq w_i^*$ implies $j \neq i$, the term $a^{\ell^*+1}bs/c_{\ell^*+1-j}$ (absent in the problem instance) does not appear in the summation of $C_{i,2}$. The other components of $\mathcal{C}$ are also well-formed. The components, however, do not have the right distribution. To obtain the proper distribution, $\mathscr{B}$ uses an algorithm $\mathsf{ReRandCT}$ that rerandomises the ciphertext. A description is provided in Section 4.4. Now $\mathscr{B}$ computes $\mathcal{C}^* \leftarrow \mathsf{ReRandCT}(\mathcal{C})$ and returns $\mathcal{C}^*$ to $\mathscr{A}$.

Guess: $\mathscr{A}$ returns its guess β' of β.

If $T = a^{\ell^*+1}bsP$, then the $\mathcal{C}^*$ will be a properly distributed ciphertext. In this case $\mathscr{B}$ plays $\mathrm{Game}_{\mathrm{real}}$. Otherwise, if $T \in_{\mathrm{U}} \mathbb{G}_T$, then $\mathcal{C}^*$ will be an encyrption of a random message and hence $\mathscr{B}$ plays $\mathrm{Game}_{\mathrm{random}}$.

The simulator returns 1 if $\beta = \beta'$ and 0 otherwise. We have

$$\begin{aligned}\mathsf{Adv}_{\mathcal{G}}^{\ell^*\text{-XBDHE}}(\mathscr{B}) &= \left|\Pr[\mathscr{B}(\mathcal{D}, e(P,P)^{a^{i+1}bs}) = 1] - \Pr[\mathscr{B}(\mathcal{D}, X_T) = 1]\right| \\ &= \left|\Pr[\beta = \beta' | T = e(P,P)^{a^{i+1}bs}] - \Pr[\beta = \beta' | T \in_{\mathrm{U}} \mathbb{G}_T]\right| \\ &= \left|\Pr[\mathscr{A} \text{ wins in } \mathrm{Game}_{real}] - \frac{1}{2}\right| \\ &= \mathsf{Adv}_{\mathrm{FE}}^{\mathsf{IND\text{-}STR\text{-}CPA}}(\mathscr{A})\end{aligned}$$

Therefore, $\varepsilon' \le \varepsilon$. $\square$

4.4 Algorithms for Re-Randomization

As mentioned in the security proof, the challenge ciphertext and responses to key extraction queries are constructed using elements given in the problem instance in a way resulting in improper distribution. The simulator must re-randomise them suitably before providing them to the adversary. To this end, we use two algorithms $\mathsf{ReRandCT}$ and $\mathsf{ReRandK}$ that re-randomise the ciphertext and keys respectively.

$\mathsf{ReRandCT}(\mathcal{C})$: This algorithm picks $s'_0, s'_1, \ldots, s'_\ell \xleftarrow{\mathrm{U}} \mathbb{Z}_p$ and modifies the ciphertext elements as shown below.

$$C_m \leftarrow C_m \cdot e(P,P)^{\alpha s'_\ell},$$

$$C_{\text{start},1} \leftarrow C_{\text{start},1} + s'_0 P, \quad C_{\text{start},2} \leftarrow C_{\text{start},2} + s'_0 H_{\text{start}},$$

For $i = 1, \ldots, \ell$,
$$C_{i,1} \leftarrow C_{i,2} + s'_i P, \quad C_{i,2} \leftarrow C_{i,2} + s'_i H_{w_i} + s'_{i-1} P_1, \quad C_{i,3} \leftarrow C_{i,3} + s'_{i-1} H_\lambda + s'_{i-1} P_1,$$

$$C_{\text{end},1} \leftarrow C_{\text{end},1} + s'_\ell P, \quad C_{\text{end},2} \leftarrow C_{\text{end},2} + s'_\ell H_{\text{end}}.$$

The new randomisers for the ciphertext will be $s_i + s'_i$ $(i = 0, \ldots, \ell)$. The string w remains the same.

$\mathsf{ReRandK}(\mathcal{SK}_{\mathcal{M}})$: Choose uniform and independent random scalars r'_{start}, for all $t \in \mathcal{T}$, r'_t and for all $q_x \in F, Z_{y_1}, Z_{y_2} \in \Gamma$, $r'_{\text{end}_{(x,y_1,y_2)}}$ from $\mathbb{Z}_p$. Also choose $D'_{x,y_1,y_2} \xleftarrow{\mathrm{U}} \mathbb{G}$ for every $(x,y) \in \mathbb{Z}_{|Q|} \times \mathbb{Z}_{|\Gamma|}$. Reconstruct components of the key as follows.

$$K_{\text{start},1} \leftarrow K_{\text{start},1} + D'_{0,0,0} + r'_{\text{start}} H_{\text{start}}, \quad K_{\text{start},2} \leftarrow K_{\text{start},2} + r'_{\text{start}} P$$

$\forall t \in \mathcal{T}$ with $t = (q_x, q_{x'}, \sigma, Z_{y_1}, Z_{y_2}, Z_{y'_1}, Z_{y'_2})$ and $\sigma \in \Sigma \cup \{\lambda\}$,
$$K_{t,1} \leftarrow K_{t,1} - D'_{x,y_1,y_2} + r'_t P_1, \quad K_{t,2} \leftarrow K_{t,2} + r'_t P, \quad K_{t,3} \leftarrow K_{t,3} + D'_{x',y'_1,y'_2} + r'_t H_\sigma,$$

$\forall q_x \in F$, $\forall Z_{y_1}, Z_{y_2} \in \Gamma$,
$$K_{\text{end}_{(x,y_1,y_2)},1} \leftarrow K_{\text{end}_{(x,y_1,y_2)},1} + D'_{x,y_1,y_2} + r'_{\text{end}_{(x,y_1,y_2)}} H_{\text{end}} ,$$
$$K_{\text{end}_{(x,y_1,y_2)},2} \leftarrow K_{\text{end}_{(x,y_1,y_2)},2} + r'_{\text{end}_{(x,y_1,y_2)}} P.$$

Similar to ciphertext re-randomisation, this algorithm does additive re-randomisation.

References

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003. Earlier version appeared in the proceedings of CRYPTO 2001.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society, 2007.

[BSW12] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: a new vision for public-key cryptography. *Commun. ACM*, 55(11):56–64, 2012.

[Coc01] Clifford Cocks. An identity-based encryption scheme based on quadratic residues. In Bahram Honary, editor, *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer, 2001.

[GGH12] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. Cryptology ePrint Archive, Report 2012/610, 2012. `http://eprint.iacr.org/`.

[GKP$^+$12] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Succinct functional encryption and applications: Reusable garbled circuits and beyond. Cryptology ePrint Archive, Report 2012/733, 2012. `http://eprint.iacr.org/`.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 89–98. ACM, 2006.

[Ham11] Mike Hamburg. Spatial encryption. Cryptology ePrint Archive, Report 2011/389, 2011. `http://eprint.iacr.org/`.

[HMU00] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 2 edition, 2000.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2008.

[LOS$^+$10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 62–91. Springer, 2010.

[OSW07] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 195–203. ACM, 2007.

[OT09] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 214–231. Springer, 2009.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 191–208. Springer, 2010.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.

[SW12] Amit Sahai and Brent Waters. Attribute-based encryption for circuits from multilinear maps. *arXiv preprint arXiv:1210.5287*, 2012.

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2011.

[Wat12] Brent Waters. Functional encryption for regular languages. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 218–235. Springer, 2012.