Towards Symmetric Functional Encryption for Regular Languages with Predicate Privacy

Fu-Kuo Tseng, Rong-Jaye Chen, and Bao-Shuh Paul Lin

National Chiao-Tung University,
No.1001, Daxue Road, Hsinchu City 300, Taiwan
{fktseng,rjchen)@cs.nctu.edu.tw; bplin@mail.nctu.edu.tw

Abstract. We present a symmetric-key predicate-only functional encryption system, SP-FE, which supports functionality for regular languages described by deterministic finite automata. In SP-FE, a data owner can encrypt a string of symbols as encrypted symbols for matching. Later, the data owner can generate predicate tokens of the transitions in a deterministic finite automaton (DFA). The server with these tokens can decrypt a sequence of encrypted symbols correctly and transfer from one state to another accordingly. If the final state belongs to the set of accept states, the server takes assigned operations or returns the corresponding encrypted data. We have proven SP-FE preserves both keyword privacy and predicate privacy through security analysis and security games. However, to achieve predicate privacy, we put bounds on the length of a string and the number of states of a DFA. Due to these restrictions, SP-FE can only capture finite languages. Finally, we present the performance analysis of SP-FE and mention possible future work.

Keywords: symmetric functional encryption, deterministic finite automaton, regular language, predicate-only scheme, predicate privacy

1 Introduction

Functional public-key encryption schemes [1] and many of their instances like attribute-based encryption schemes [2, 3, 4] and predicate encryption schemes [5, 6, 7] were devised to support expressive search predicates on encrypted data. However, in all of these schemes, search predicates involve only a *fixed* number of keywords from the predefined keyword universe. Processing a string of encrypted symbols representing a keyword is essential for the predicates of certain regular languages for lexical analysis and pattern matching.

However, the predicate tokens in functional encryption schemes reveal the content of the search predicates because encryption does not require a private key in the public-key setting. Adversaries may encrypt the keywords of their choices and check the ciphertexts with the delegated predicate token to learn whether the chosen keywords satisfy the search predicate encoded in the predicate token. Therefore, predicate privacy is inherently impossible to achieve in the public-key setting. Researchers started focusing on the symmetric-key setting

for predicate privacy with keyword-based search predicates [8, 9, 10]. Functional encryption for regular languages, a type of symbol-based search predicates, was considered in [11] and [12], while functional encryption for regular languages with additional *keyword and predicate privacy* is still an open problem.

Our Contributions. We propose a symmetric-key predicate-only functional encryption scheme, SP-FE, supporting predicates of deterministic finite automata (DFA). We make use of Yoshino et al. scheme [10] by encrypting each symbol in the plaintext as an encrypted symbol and each symbol set of a transition as a predicate token. However, direct transformation cannot achieve both plaintext and predicate privacy because some information may reveal to the adversary: (1) The length of a plaintext, n_w, (2) the number of states of a DFA, n_Q, (3) the number of accept states, $|F|$, (4) the number of transitions, $|\delta|$, and (5) the path of the transition. Thus, we systematically add special symbols among ordinary symbols in the plaintext, while inserting dummy states, transitions and shuffling states of a DFA. These designs guarantee the adversaries cannot gain extra advantages in the challenge games. However, to achieve predicate privacy, we put bounds on the length of a string and the number of (accept) states of a DFA, thus SP-FE can only capture finite languages.

Related Works. Functional encryption schemes [1] are non-interactive public-key encryption schemes where anyone possessing a secret key sk_f can compute a function $f(x)$ of a value x from the encryption $\texttt{Enc}(x)$ without learning any other information about x. However, the predicate tokens in these functional encryption schemes may reveal the content of the underlying search predicates because encryption does not require a private key in the public-key setting. Thus predicate privacy is inherently impossible to achieve in the public-key setting. Shen et al. [9] was the first to consider predicate privacy in the symmetric-key setting. Blundo et al. [8] used the assumptions related to linear split secret sharing, while Yoshino et al. [10] further enhanced the efficiency by prime-order group instantiation. However, in all of these schemes, search predicates involve only a fixed number of keywords in the keyword univserse. Processing a string of searchable symbols, is essential for the search predicates of regular languages. The functional encryption for regular languages was devised in the public-key setting with message privacy only [11]. The functional encryption for regular languages with extra *keyword* and *predicate privacy* is still an open problem.

2 Background and Preliminary

This section presents the background and preliminary of SP-FE.

2.1 Deterministic Finite Automata and Regular Languages

A deterministic finite automaton (DFA) is a finite state machine that accepts or rejects finite strings of symbols. A DFA M is a quintuple $(Q, \Sigma, \delta, q_0, F)$ where (1) Q is a finite set of states, (2) Σ is the input alphabet, a finite set of symbols, (3) $\delta : Q \times \Sigma \to Q$ is the transition function, where $(q, q', \sigma) \in \delta$ iff $\delta(q, \sigma) = q'$, (4) q_0 is an initial state, and (5) F is the set of final states, a subset of Q. $\square$

Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, if M accepts an string $w = w_1 w_2, \cdots, w_n \in \Sigma^n$, there exists a sequence of states, a transition path, $r = r_0, r_1, ..., r_n \in Q^n$, where (1) $r_0 = q_0$, (2) $\delta(r_i, w_{i+1}) = r_{i+1}$, for $0 \leq i \leq n-1$, and (3) $r_n \in F$. A regular language is also defined as a language recognized by a DFA.

2.2 Definitions and Security Model

A symmetric-key predicate-only functional encryption scheme for DFA-based predicates consists of four algorithms: Setup, Encrypt, KeyGen, and Decrypt. In addition to a security parameter, the setup algorithm takes as input a alphabet Σ and a special alphabet Σ'. The algorithms are described as follows.

- $Setup(1^\lambda, \Sigma, \Sigma')$: It takes a security parameter 1^λ as input and outputs public parameters and a secret key SK. Public parameters include two parameters to decide the maximum length of an input string, N_w, the maximum number of states, N_Q, and the maximum number of transitions in a DFA, $N_\delta = N_Q^2$.
- $Encrypt(SK, w)$: It takes a secret key SK and a string (of symbols) w, where $|w| \leq N_w/2$, and outputs a ciphertext CT, a string of encrypted symbols.
- $KeyGen(SK, M{=}(Q, \Sigma, \delta, q_0, F))$: It takes a secret key SK and a DFA M as input, where $|Q| \leq N_Q/2$ and outputs a TK, a string of encrypted transitions.
- $Decrypt(TK, CT)$: It takes a token TK and a ciphertext CT as input, and outputs either '1' ('Accept') or '0' ('Reject') indicating that the result of the DFA M encoded in TK on the input w encrypted in CT. □

Security Model. The selective game-based security is considered in SK-FE.

- **Setup:** The challenger C runs the setup algorithm and gives public parameters to the adversary $\mathcal{A}$. $\mathcal{A}$ outputs a bit $d \in \{0, 1\}$: if $d = 0$, $\mathcal{A}$ takes up a ciphertext challenge and outputs two plaintext w_0 and w_1. Otherwise, $\mathcal{A}$ takes up a token challenge and outputs two description of DFA M_0 and M_1.

- **Phase 1:** $\mathcal{A}$ adaptively outputs one of the following two queries. In a ciphertext challenge, $\mathcal{A}$ issues ith ciphertext query by requesting for the ciphertext CT^i of w^i. C responds with $CT^i \leftarrow Encrypt(SK, w^i)$. Also, $\mathcal{A}$ issues jth token query by requesting a DFA M^j with the restriction that M^j accepts or rejects both w_0 and w_1 with the same number of accept states in the transition paths. C responds with $TK^j \leftarrow KeyGen(SK, M^j)$. In a token challenge, $\mathcal{A}$ issues ith ciphertext query by requesting for a string of ciphertext CT^i of w^i with the restriction that w^i is accepted or rejected by both M_0 and M_1 with the same number of accept states in the transition paths. C responds with $CT^i \leftarrow Encrypt(SK, w^i)$. Also, $\mathcal{A}$ issues jth token query by requesting for a DFA M^j. C responds with $TK^j \leftarrow KeyGen(SK, M^j)$. The restrictions is to ensure the challenge is not trivial.

- **Challenge:** The challenger C flips a random coin $b \in \{0, 1\}$. If $\mathcal{A}$ has chosen the ciphertext challenge, C gives $CT_b \leftarrow Encrypt(SK, w_b)$ to $\mathcal{A}$; otherwise ($\mathcal{A}$ has chosen the token challenge), C gives $TK_b \leftarrow KeyGen(SK, M_b)$ to $\mathcal{A}$.

- **Phase 2:** $\mathcal{A}$ continues to query CT^i and TK^j as in Phase 1.

- **Guess:** $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ of b. The advantage of an adversary $\mathcal{A}$ in this game is defined as $Pr[b' = b] - \frac{1}{2}$.

Definition 1. *A symmetric-key predicate-only functional encryption scheme for DFA-type predicates is token indistinguishable if all polynomial-time adversaries have at most a negligible advantage in winning the token challenge game. This property guarantees predicate privacy.*

Definition 2. *A symmetric-key predicate-only functional encryption scheme for DFA-type predicates is ciphertext indistinguishable if all polynomial-time adversaries have at most a negligible advantage in winning the ciphertext challenge game. This property guarantees keyword privacy.*

2.3 Notation

Σ is a set of ordinary symbols used to form a keyword/plaintext w, while Σ' is a set of special symbols randomly added into w to form w'. The special symbols cannot be specified in w. The union of these two sets forms Σ''. Each symbol $w_i \in \Sigma''$ has a unique index s_i. The sizes of these three sets are σ, σ', and σ'' respectively. n_w denotes the length of w, while N_w denotes the maximum length of w', where $n_w \leq \frac{1}{2}N_w$. In addition, there are $\lceil \frac{1}{2}N_w \rceil$ groups of special symbols, whose size is from 1 to $\lceil \frac{1}{2}N_w \rceil$. A group of special symbols are added as a set in a predefined circular order starting with any one of the symbols in this group.

A predicate DFA M is denoted as $(\Sigma, Q, \delta, q_0, F)$. Redundant states are chosen from Q to form Q' and from F to form F', while duplicated transitions are included in δ to form δ'. Q' and δ' are randomized to form Q'' and δ''. The sizes of Q, Q' and Q'' are n_Q, n'_Q and $n''_Q = N_Q$ respectively, where $N_Q \geq 2n_Q$. The states in Q are marked from 0 to $n_Q - 1$, thus the number of transition in δ is $n_\delta = n_Q^2$. N_Q denotes the maximum number of states, thus the maximum number of transitions N_δ is N_Q^2. Note that we further require $|F| \leq N_Q/4$ and $|Q| \leq N_Q/2$. q_0 and F' are randomized into q'_0 and F''. δ and its matrix representation A_δ can be converted. If $A_\delta[x][y]$ is a symbol set W, where $W \subseteq \Sigma''$ and $x, y \in Q''$, there are (x, y, w_i) transitions in δ, where $w_i \in W$.

2.4 Complexity Assumption

Given a bilinear 3-factor-based composite-order group generator $\mathcal{G}$, output three groups $\mathbb{G}_i$ of distinct prime order p_i for $i = 1, 2, 3$ by the experiment:

1. $(p_1, p_2, p_3, \mathbb{G}, \mathbb{G}_T, \hat{e}) \leftarrow \mathcal{G}(1^\lambda)$,
2. $N \leftarrow p_1p_2p_3$, $g_1 \xleftarrow{R} \mathbb{G}_1$, $g_2 \xleftarrow{R} \mathbb{G}_2$, $g_3 \xleftarrow{R} \mathbb{G}_3$,
3. $P \leftarrow (N, \mathbb{G}, \mathbb{G}_T, \hat{e})$,
4. $D \leftarrow (g_1, g_1^{a_1}, g_2, g_2^{b_1}, g_3^{c_1}, g_3^{c_2 d}, g_3^d, g_3^{d^2}, g_1^{a_2} g_3^{c_1 d})$, where $a_1, a_2 \xleftarrow{R} \mathbb{Z}_{p_1}$, $b_1 \xleftarrow{R} \mathbb{Z}_{p_2}$, and $c_1, c, d \xleftarrow{R} \mathbb{Z}_{p_3}$, and
5. $T_0 \leftarrow g_1^{a_3} g_3^{c_2}$, $T_1 \leftarrow g_1^{a_3} g_2^{b_2} g_3^{c_2}$, where $a_3 \xleftarrow{R} \mathbb{Z}_{p_1}$ and $b_2 \xleftarrow{R} \mathbb{Z}_{p_2}$.

The advantage of an adversary $\mathcal{A}$ in distinguishing T_0 from T_1 with the parameters (P, D) is defined as $Adv_\mathcal{A} := |\Pr[\mathcal{A}(P, D, T_0) = 1] - \Pr[\mathcal{A}(P, D, T_1) = 1]|$.

Definition 3. The above complexity assumption holds for any polynomial-time adversary $\mathcal{A}$ if $Adv_\mathcal{A}$ is negligible [10].

Procedure: SymbolSetToVector(W, *mode*) [6]
Input: W, where $W \subseteq \Sigma''$, $|\Sigma''| = \sigma''$; *mode* =0: TK, *mode* =1: CT;
Each symbol $w_i \in \Sigma''$ has a unique index s_i
Output: $\boldsymbol{v}_W$

---

if (W **is** $\emptyset$) **then** $\boldsymbol{v}_W = (a_{\sigma''}, a_{\sigma''-1}, \ldots, a_0) = (0, 0, \cdots, 0)$
else if (mode **is** 0) **then**
$\boldsymbol{v}_W = (a_{\sigma''}, a_{\sigma''-1}, \ldots, a_{d+1}, a_d, a_{d-1}, \ldots, a_0)$, where
$f(x) = \prod_1^d (x - s_i) = a_d x^d + a_{d-1} x^{d-1} + \ldots + a_0$ and $a_{\sigma''} = \ldots = a_{d+1} = 0$.
else (mode **is** 1) **then**
$\boldsymbol{v}_W = (s_i^{\sigma''} \bmod N, \ldots, s_i^0 \bmod N) = (a_{\sigma''}, a_{\sigma''-1}, \ldots, a_0)$, where N is the order of the groups $\mathbb{G}$ and $\mathbb{G}_T$
return $\boldsymbol{v}_W$

Fig. 1. The procedure 'SymbolSetToVector'

2.5 The Building Block

The scheme by Yoshino et al. [10] provides a good starting point to construct SP-FE. It is a keyword-based predicate-only predicate encryption scheme.

- `IPE.Setup`(1^λ): It takes a security parameter 1^λ as input and outputs public parameters and a secret key SK.
- `IPE.Encrypt`(SK, x): It takes a secret key SK and a plaintext $x \in \Sigma$ and outputs a ciphertext CT.
- `IPE.GenToken`(SK, y): It takes a secret key SK and a description of predicate y as input and outputs a token TK.
- `IPE.Check`(TK, CT): It takes a token TK and a ciphertext CT as input and outputs either '1' ('Accept') or '0' ('Reject') indicating the result of the predicate y encoded in TK on the input x encrypted into CT. □

Note that disjunctive predicates are used to protect the input symbols of a transition in SK-FE. The procedure **SymbolSetToVector** is to generate the vector of a input string and that of a symbol set for a transaction (See Fig. 1).

3 SP-FE Construction

We provide main procedures of SP-FE construction. Following that, we present detailed algorithms with comprehensive explanation.

3.1 Main Procedures

We make use of Yoshino *et al.* scheme [10] by encrypting each symbol in the plaintext as an encrypted symbol and each symbol set for a transition as a predicate token. However, direct transformation cannot achieve both keyword and predicate privacy because the information can be revealed to the adversary: (1) The length of a plaintext, (2) the number of states in a DFA, (3) the number of accept states, (4) the number of transitions, and (5) the transition path. Thus, **addSpecialSymbols** adds special symbols to the plaintext, while **addStatesTransitions** inserts dummy states, transitions and shuffs states of a DFA accordingly to make sure the same language is accepted (See Fig. 2 and Fig. 4).

Procedure: addSpecialSymbols(w)
Input: w, where $w = (w_1, \ldots, w_{n_w}), w_i \in \Sigma, n_w \leq \ell$
Output: w', where $w' = (w'_1, \ldots, w'_{N_w}), w'_i \in \Sigma'', N_w = 2\ell$

Repeat 1. and 2. until $n_w = N_w$.

1. Set $pos \xleftarrow{R} \mathbb{Z}_{n_w+1}$ and $k \xleftarrow{R} \mathbb{Z}_{N_w - n_w}$
2. Insert the $(k+1)th$ symbol group at position pos with a pre-defined circular ordering starting from one of the symbols in the group. Set $n_w = n_w + (k+1)$.

return $w' = (w_1, w_2, \ldots, w_{N_w})$

Fig. 2. The procedure 'addSpecialSymbol'

a >< ∧a ⊣⊢ a | ♯aa⌞⌜⌝⌟a | aa <⊢⊣ ∧ > a | aa⌟⌞⌜♯⌝a | a ⊣⊢ a >< ∧a | aa♯ < ∧♯ > a | ♯aa ⊢⊣ ♯♯a

Fig. 3. The procedure 'addSpecialSymbol'('aaa') and its seven possible outputs

Procedure: addStatesTransitions(M)
Input: $M{=}(Q, \Sigma, \delta, q_0, F), n_Q \leq \ell$, where
$|\delta| = n_\delta, \delta = \{(u^j, v^j, w_k)\}_{j=1}^{n_\delta}; u^j, v^j \in Q$ and $w_k \in \Sigma$
Output: $M''{=}(Q'', \Sigma'', \delta'', q'_0, F'')$, where $N_Q = 2\ell$ and $|\delta''| = N_\delta = N_Q^2$

1. *Add Random Symbols*: For each row i in a DFA, each of the symbols in Σ'' should appear once and only once. If a symbol w_i of a group of special symbols of size d does not appear in the row i,
   (a) Prepare the sequence $w_i, w'_1, w'_2, \cdots, w'_d$ starting with w_i and a sequence of states $i, t_1, t_2, \cdots, t_d$, where w_i does not appear in row i and w'_j does not appear in row t_j for $1 \leq j \leq d$ and $t_j \in Q$
   (b) Include the transitions (i, t_1, w_i), (t_j, t_{j+1}, w'_j) for $1 \leq j \leq d-1$, and (t_d, i, w'_d) into δ to form δ'.
2. *Add Random States, Add Final States and Transitions*:
   (a) Randomly duplicate $(\frac{\ell}{2} - |F|)$ states from F to form F'. The new state creates a new column and copies the row of its original state as its row.
   (b) Randomly duplicate $\frac{\ell}{2}$ states from the $\frac{\ell}{2}$ states in 1. together with F' to form Q'. Denote S_i as the set of equivalent states of the state i, where $S_i \subseteq Q'$, $\cup_{i=1}^{|Q|} S_i = Q'$, and $S_i \cap S_j = \emptyset$ for any two sets.
   There are extra $\ell^2 - n_q^2$ transitions added into δ to form δ'.
3. *Shuffle Symbols within Equivalent Set*: For each row i, the transition symbols are shuffled among the columns of the equivalent states to form δ''.
4. *Shuffle States*: Randomly choose two states Q_i, Q_j. Exchange ith row with jth row, and ith column with jth column to form Q'' and δ''. Set one state in S_{q_0} as a starting state q'_0, while set F'' as the final states after exchange.

return $M''{=}(Q'', \Sigma'', \delta'', q'_0, F'')$, where $\delta''{=}\{(u^j, v^j, w_k)\}_{j=1}^{N_\delta}$

Fig. 4. The procedure 'addStatesTransitions'

Example. In Fig. 1, ℓ is set as four. There are four groups of special symbols denoted as (1)'♯', (2)'⊢' and '⊣', (3)'<', '∧' and '>', and (4) '⌜', '⌝', '⌟' and '⌞'. We have d ordered sequences of a group of size d. For example, to insert the symbols in the third group, one of the three sequences can be chosen: '< ∧ >', '∧ ><', and '>< ∧'. In addition, one group of symbols can be nested in the other group of symbols like in the third, fourth and sixth column in Fig. 1. After a group of symbols are consumed by a DFA, their effects will be canceled out.

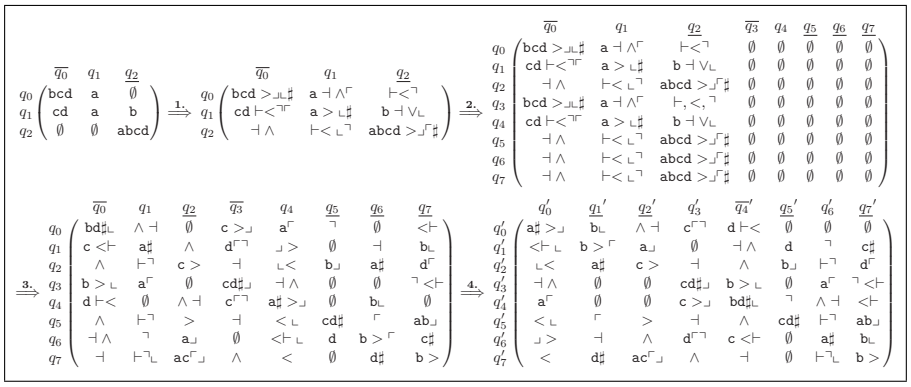

Fig. 5. The procedure 'addStatesTransitions' processing 'containing substring ab'

Example. In Fig. 5, ℓ is set as four and $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$. For each row, every symbols in Σ'' should appear once and only once. In addition, the input DFA has specified all the symbols in Σ for each row. To fill in a special symbol of a group, there is a transition path from state i back to state i again after consuming the ordered circular sequence starting from this symbol. Take symbol '$>$' as example, the sequence is '$>$', '$<$' and then '$\wedge$'. There is a transition path: $q_0 \overset{>}{\Rightarrow} q_0 \overset{<}{\Rightarrow} q_2 \overset{\wedge}{\Rightarrow} q_0$ and there are $(q_0, q_0, >)$, $(q_0, q_2, <)$, and $(q_2, q_0, \wedge)$ transitions in δ. The next step is to duplicate the set of accept states so that $F' = \ell$ and duplicate the other states so that $Q'' = 2\ell$. Therefore, there are three equivalent sets: $S_{q_0} = \{q_0, q_3\}$, $S_{q_1} = \{q_1, q_4\}$, and $S_{q_2} = \{q_2, q_5, q_6, q_7\}$. For the third step, the symbols of one equivalent set in one row can be redistributed. Take q_2 row as example. '$\dashv$' is moved from q_0-column into q_3-column, while '$\llcorner$' is moved from q_1-column, into q_4-column. Finally, exchange state 0 with state 4 and state 1 with 6 by interchanging q_0-row with q_4-row, q_0-column with q_4-column, q_1-row with q_6-row, and q_1-column with q_6-column. Set the start state q'_0 from $S'_{q_0} = \{q'_3, q'_4\}$ as q'_4 and hide the others. Set the set of final states F'' as $\{q'_1, q'_2, q'_5, q'_7\}$.

3.2 Main Algorithms

SP-FE consists of four probabilistic polynomial-time algorithms (See Fig. 6). In SP-FE.Setup, the user executes `IPE.Setup` to obtain a SK and system parameters. In `SP-FE.Encrypt`, the user executes **addSpecialSymbols** on input w to produce w'. Then the user calls to **symbolSetToVector** and `IPE.Encrypt` for each of the symbols in w' to produce a CT. In SP-FE.GenToken, the user executes **addStatesTransitions** on the search predicate M to produce M''. Then the user calls to **symbolSetToVector** and `IPE.GenToken` for each of the transitions in δ'' to produce a token TK. In SP-FE.Decrypt, the server executes `IPE.Check` on input CT_i and $TK_{q_c,j}$ to test a transition $(q_c, j, TK_{q_c,j})$ in δ''. The server obtains the next state j if $\texttt{IPE.Check}(CT_i, TK_{q_c,j})$ returns '1' and set the current state q_c as j. The server continues to check CT_{i+1} with the transitions in δ'' starting with q_c. If the final state q_c is in F'' after checking CT_{N_w}, the plaintext w' satisfies the DFA in M''.

SP-FE.Setup(1^λ):
$SK \leftarrow$ `IPE.Setup`$(1^\lambda, \Sigma, \Sigma')$
Set ℓ, where $N_w = 2\ell, N_Q = 2\ell, N_\delta = N_Q^2$
return SK

SP-FE.GenToken$(SK,\ M{=}(\Sigma, Q, \delta, q_0, F))$:
$M'' \leftarrow$ **addStatesTransitions**(M), where
$M''{=}(\Sigma'', Q'', \delta'', q_0', F''), |Q''| = N_Q, |\delta''| = N_Q^2$
for$(r = 1$ to $N_Q)$ **do**
for$(c = 1$ to $N_Q)$ **do**
$(r, c, \boldsymbol{y}_{r,c}) \leftarrow$ **symbolSetToVector**$(A_{\delta''}[r][c],0)$
$TK = TK \cup TK_{r,c}$, where
$TK_{r,c} = (r, c,$`IPE.GenToken`$(SK,\ \boldsymbol{y}_j))$
end of for
end of for
return TK

SP-FE.Encrypt$(SK, w = w_1, \ldots, w_{n_w})$:
$w' \leftarrow$ **addSpecialSymbols**(w), where $|w'| = N_w$
for $(i = 1$ to $N_w)$ **do**
$\boldsymbol{x}_i \leftarrow$ **symbolSetToVector**$(w_i', 1)$
$CT = CT \cup CT_i =$ `IPE.Encrypt`$(\boldsymbol{x}_i)$
end of for
return $CT = \{CT_i\}_{i=1}^{N_w}$

SP-FE.Decrypt$(CT,\ M''{=}(Q'', \Sigma'', \delta'', q_0', F''))$:
$q_c = q_0'$, where q_c is current state
for $(i = 1$ to $N_w)$ **do**
for all $(q_c, j, TK_{q_c,j}) \in \delta''$, where $j \in Q''$ **do**
if (`IPE.Check`$(CT_i, TK_{q_c,j}) == 1)$ **then**
$q_c = j$, ***break inner for-loop***
end of if
end for
end for
if $(q_c \in F'')$ **then return** 1
else return 0

Fig. 6. The main construction SK-FE

3.3 Correctness

If a keyword w is accepted (or rejected) by a DFA M, the corresponding w' is accepted (rejected)by the DFA M''. In **addSpecialSymbols**, one group of special symbols (with predefined order) can be nested in the other group of symbols. On the other hand, a group of special symbols are filled in a DFA is the same order. After a group of the special symbols between two ordinary symbols are consumed by a DFA, their effects are canceled out, namely, the same state is reached as if no special symbols are inserted.

4 Analysis

We describe a sequence of hybrid security games to demonstrate that SP-FE achieves both keyword privacy and predicate privacy.

4.1 Security Analysis

Proof Sketch. The proof uses a sequence of hybrid games where a challenge token is encrypted with one vector in the first subsystem and with another vector in the second subsystem. Let $(\boldsymbol{w}, \boldsymbol{z})$ denote a token encrypted by vector $\boldsymbol{w}$ in the first subsystem and by vector $\boldsymbol{z}$ in the second subsystem. Try to prove the challenge token associated with $\boldsymbol{w}$ corresponding to $(\boldsymbol{w}, \boldsymbol{w})$ is indistinguishable from that associated with $\boldsymbol{z}$ corresponding to $(\boldsymbol{z}, \boldsymbol{z})$. A sequence of hybrid games demonstrates $(\boldsymbol{w}, \boldsymbol{w}) \simeq (\boldsymbol{w}, \boldsymbol{0}) \simeq (\boldsymbol{w}, \boldsymbol{z}) \simeq (\boldsymbol{0}, \boldsymbol{z}) \simeq (\boldsymbol{z}, \boldsymbol{z})$ (See Fig. 7). As the space is limited, we give these proofs in the full version of our paper [13]. In addition, n_w, n_Q, F and δ are hidden from the adversary. The distribution of two paths of the transition is computationally indistinguishable because there are at least $N_w/2$ symbols added into the input string of length at most $N_w/2$.

Game 1. The token encrypted by vector $(\boldsymbol{w}, \boldsymbol{w})$.

$$\{\mathrm{TK}_j\}_{j=1}^{N_\delta} = \left\{ \begin{array}{l} (u^j, v^j), \{g_1^{V_{1,i}} g_2^{\beta_1 w_i} g_3^{Tr_{1,i}^j}\}_{i=1}^{\sigma''+1}, \\ \{g_1^{V_{2,i}} g_2^{\beta_2 w_i} g_3^{Tr_{2,i}^j}\}_{i=1}^{\sigma''+1}, \Pi_{i=1}^{\sigma''+1} g_1^{-V_{1,i}q_{1,i}^j - V_{2,i}q_{1,i}^j} g_2^{V_1} g_3^{V_2}, g_3^T \end{array} \right\}_{j=1}^{N_\delta}$$

Game 2. The token encrypted by vector $(\boldsymbol{w}, \boldsymbol{0})$.

$$\{\mathrm{TK}_j\}_{j=1}^{N_\delta} = \left\{ \begin{array}{l} (u^j, v^j), \{g_1^{V_{1,i}} g_2^{\beta_1 w_i} g_3^{Tr_{1,i}^j}\}_{i=1}^{\sigma''+1}, \\ \underline{\{g_1^{V_{2,i}} g_3^{Tr_{2,i}^j}\}_{i=1}^{\sigma''+1}}, \Pi_{i=1}^{\sigma''+1} g_1^{-V_{1,i}q_{1,i}^j - V_{2,i}q_{1,i}^j} g_2^{V_1} g_3^{V_2}, g_3^T \end{array} \right\}_{j=1}^{N_\delta}$$

Game 3. The token encrypted by vector $(\boldsymbol{w}, \boldsymbol{z})$.

$$\{\mathrm{TK}_j\}_{j=1}^{N_\delta} = \left\{ \begin{array}{l} (u^j, v^j), \{g_1^{V_{1,i}} g_2^{\beta_1 w_i} g_3^{Tr_{1,i}^j}\}_{i=1}^{\sigma''+1}, \\ \underline{\{g_1^{V_{2,i}} g_2^{\beta_2 z_i} g_3^{Tr_{2,i}^j}\}_{i=1}^{\sigma''+1}}, \Pi_{i=1}^{\sigma''+1} g_1^{-V_{1,i}q_{1,i}^j - V_{2,i}q_{1,i}^j} g_2^{V_1} g_3^{V_2}, g_3^T \end{array} \right\}_{j=1}^{N_\delta}$$

Game 4. The token encrypted by vector $(\boldsymbol{0}, \boldsymbol{z})$.

$$\{\mathrm{TK}_j\}_{j=1}^{N_\delta} = \left\{ \begin{array}{l} (u^j, v^j), \{g_1^{V_{1,i}} g_3^{Tr_{1,i}^j}\}_{i=1}^{\sigma''+1}, \\ \{g_1^{V_{2,i}} g_2^{\beta_2 z_i} g_3^{Tr_{2,i}^j}\}_{i=1}^{\sigma''+1}, \Pi_{i=1}^{\sigma''+1} g_1^{-V_{1,i}q_{1,i}^j - V_{2,i}q_{1,i}^j} g_2^{V_1} g_3^{V_2}, g_3^T \end{array} \right\}_{j=1}^{N_\delta}$$

Game 5. The token encrypted by vector $(\boldsymbol{z}, \boldsymbol{z})$.

$$\{\mathrm{TK}_j\}_{j=1}^{N_\delta} = \left\{ \begin{array}{l} (u^j, v^j), \underline{\{g_1^{V_{1,i}} g_2^{T\beta_1 z_i} g_3^{r_{1,i}^j}\}_{i=1}^{\sigma''+1}}, \\ \{g_1^{V_{2,i}} g_2^{\beta_2 z_i} g_3^{Tr_{2,i}^j}\}_{i=1}^{\sigma''+1}, \Pi_{i=1}^{\sigma''+1} g_1^{-V_{1,i}q_{1,i}^j - V_{2,i}q_{1,i}^j} g_2^{V_1} g_3^{V_2}, g_3^T \end{array} \right\}_{j=1}^{N_\delta}$$

Fig. 7. The procedure 'a sequence of hybrid games'

Theorem 1. *If $\mathcal{G}$ satisfies Assumption* 1*, SP-FE is token indistinguishable.*

Theorem 2. *If $\mathcal{G}$ satisfies Assumption* 1*, SP-FE is ciphertext indistinguishable.*

Corollary 1. *If $\mathcal{G}$ satisfies Assumption* 1*, SP-FE is selective secure, namely, SP-FE achieves both keyword privacy and predicate privacy.*

4.2 Performance Analysis

The performance of SP-FE is as follows: SP-FE.Encrypt requires N_w number of `SK-PE.Encrypt`. SP-FE.GenToken costs N_Q^2 number of `SK-PE.GenToken`. SP-FE.Decrypt takes $\frac{1}{2}N_Q$ number of `SK-PE.Check` in average for each transition. For the performance of `SK-PE`, `SK-PE.Encrypt` $(8\sigma''+2)\cdot T_{add}+(13\sigma''+3)\cdot T_{sm}$, while `SK-PE.GenToken` takes $(8\sigma''+2)\cdot T_{add}+(13\sigma''+3)\cdot T_{sm}$. `SK-PE.Check` takes $(2\sigma''+2)\cdot T_{pairing}$. T_{add}, T_{sm} and $T_{pairing}$ denote the time for the point addition in $\mathbb{G}$, the scaler multiplication in $\mathbb{G}$ and the embedded pairing function. On the other hand, a plaintext w of length n_w is encrypted as N_w symbols. The size of the ciphertext is $N_w \cdot (2\sigma''+2) \cdot |\mathbb{G}|$, while that of the token is $N_\delta \cdot (2\sigma''+2) \cdot |\mathbb{G}|$ plus the description of the DFA, where $|\mathbb{G}|$ is the size of the element in $\mathbb{G}$.

5 Conclusions

In this paper, we proposed a symmetric-key predicate-only functional encryption scheme SP-FE, which supports functionality for regular languages. SP-FE is

proven to guarantee keyword privacy and predicate privacy. In addition, SP-FE can be extended to a full-fledged functional encryption scheme by the technique from [6] to further manage messages. For future work, we would like to relax the restrictions in SP-FE to support predicates of more expressive languages.

References

[1] Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In Ishai, Y., ed.: Theory of Cryptography. Volume 6597 of LNCS. Springer Berlin Heidelberg (2011) 253–273

[2] Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: Proceedings of the 2007 IEEE Symposium on Security and Privacy. SP '07, Washington, DC, USA, IEEE Computer Society (2007) 321–334

[3] Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM conference on Computer and communications security. CCS '06, New York, NY, USA, ACM (2006) 89–98

[4] Ostrovsky, R., Sahai, A., Waters, B.: Attribute-based encryption with non-monotonic access structures. In: Proceedings of the 14th ACM conference on Computer and communications security. CCS '07, New York, NY, USA, ACM (2007) 195–203

[5] Caro, A., Iovino, V., Persiano, G.: Fully secure hidden vector encryption. In Abdalla, M., Lange, T., eds.: Pairing-Based Cryptography Pairing 2012. Volume 7708 of LNCS. Springer Berlin Heidelberg (2013) 102–121

[6] Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. Advances in Cryptology–EUROCRYPT 2008 (2008) 146–162

[7] Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Gilbert, H., ed.: Advances in Cryptology EUROCRYPT 2010. Volume 6110 of LNCS. Springer Berlin Heidelberg (2010) 62–91

[8] Blundo, C., Iovino, V., Persiano, G.: Predicate encryption with partial public keys. In Heng, S.H., Wright, R., Goi, B.M., eds.: Cryptology and Network Security. Volume 6467 of LNCS. Springer Berlin Heidelberg (2010) 298–313

[9] Shen, E., Shi, E., Waters, B.: Predicate privacy in encryption systems. In Reingold, O., ed.: Theory of Cryptography. Volume 5444 of LNCS. Springer Berlin Heidelberg (2009) 457–473

[10] Yoshino, M., Kunihiro, N., Naganuma, K., Sato, H.: Symmetric inner-product predicate encryption based on three groups. In Takagi, T., Wang, G., Qin, Z., Jiang, S., Yu, Y., eds.: Provable Security. Volume 7496 of LNCS. Springer Berlin Heidelberg (2012) 215–234

[11] Waters, B.: Functional encryption for regular languages. In Safavi-Naini, R., Canetti, R., eds.: Advances in Cryptology CRYPTO 2012. Volume 7417 of LNCS. Springer Berlin Heidelberg (2012) 218–235

[12] Goldwasser, S., Kalai, Y., Popa, R., Vaikuntanathan, V., Zeldovich, N.: How to run turing machines on encrypted data. In Canetti, R., Garay, J., eds.: Advances in Cryptology CRYPTO 2013. Volume 8043 of LNCS. Springer Berlin Heidelberg (2013) 536–553

[13] Tseng, F.K., Chen, R.J., Lin, B.S.P.: Towards symmetric functional encryption for regular languages with predicate privacy. Cryptology ePrint Archive, Report 2014/407 (2014) `http://eprint.iacr.org/`.

6 Sequence of Hybrid Games

This section proves the proposed SP-FE satisfies Definition 2 (token indistinguishable) by a sequence of hybrid games from Game 1 to Game 5. Due to space limit, we give these proofs in the full version of our paper [13].

Lemma 1. *If $\mathcal{G}$ satisfies Assumption* 1*, Game* 1 *and Game* 2 *are computationally indistinguishable.*

Proof: Simulator $\mathcal{B}$ tries to break Assumption 1 by an adversary $\mathcal{A}$ trying to distinguish Game 1 from Game 2. Simulator $\mathcal{B}$ is given an instance of Assumption 1: the public parameters $(N, \mathbb{G}, \mathbb{G}_T, \hat{e})$, $(g_1, g_1^{a_1}, g_2, g_2^{b_1}, g_3^{c_1}, g_3^{c_2 d}, g_3^{d}, g_3^{d^2}, g_1^{a_2}, g_3^{c_1 d})$ and $a_1, a_2, a_3 \xleftarrow{R} \mathbb{Z}_{p_1}$, $b_1, b_2 \xleftarrow{R} \mathbb{Z}_{p_2}$, $c_1, c_2, d \xleftarrow{R} \mathbb{Z}_{p_3}$, $g_1 \xleftarrow{R} \mathbb{G}_1$, $g_2 \xleftarrow{R} \mathbb{G}_2$ and $g_3 \xleftarrow{R} \mathbb{G}_3$. Also, generate a random bit $\mathrm{b} \in \{0,1\}$, and set $T_b = g_1^{a_3} g_2^{\mathrm{b} b_2} g_3^{c_2}$.

- **Setup:** $\mathcal{A}$ is given the public parameters and outputs two challenges $M_1 = (\Sigma, Q_1, \delta_1 = \{(u^j, v^j, W)\}_{j=1}^{n_\delta}, q_{0,1}, F_1)$ and $M_2 = (\Sigma, Q_2, \delta_2 = \{(u^j, v^j), W\}_{j=1}^{n_\delta}, q_{0,2}, F_2)$. $\mathcal{B}$ converts M_1 into M_1' by **addStatesTransitions** and transforms δ' into $\{(u^j, v^j), (\boldsymbol{w}_j)\}_{j=1}^{N_\delta}$ by **symbolSetToVector**. $\mathcal{B}$ performs the same procedures on M_2 to obtain $\{(u^j, v^j), (\boldsymbol{z}_j)\}_{j=1}^{N_\delta}$. $\mathcal{B}$ sends $\{\boldsymbol{w}_j\}_{j=1}^{N_\delta}$ and $\{\boldsymbol{z}_j\}_{j=1}^{N_\delta}$ to the challenger $\mathcal{C}$. Set $\{\{q_{1,i}^j, q_{2,i}^j\}_{i=1}^{\sigma''+1}, \{r_{1,i}^j\}_{i=1}^{\sigma''+1} \{r_{2,i}'^j\}_{i=1}^{\sigma''+1}\}_{j=1}^{N_\delta}$ randomly from $\mathbb{Z}_N$.

- **Phase 1:** $\mathcal{A}$ adaptively outputs one of the two queries.

(1) Token Query. $\mathcal{B}$ receives a predicate $M=(\Sigma, Q, \delta=\{(u^j, v^j, W)\}_{j=1}^{n_\delta}, q_0, F)$ from $\mathcal{A}$. $\mathcal{B}$ turns M into $M''=(\Sigma'', Q'', \delta''=\{(u^j, v^j), W\}_{j=1}^{N_\delta}, q_0', F'')$ by **addStatesTransitions** and turns δ' into $\{(u^j, v^j), (\boldsymbol{y}_j)\}_{j=1}^{N_\delta}$ by **symbolSetToVector**. $\mathcal{B}$ randomly sets $T', \beta_1, \beta_2, \{V_{1,i}'\}_{i=1}^{\sigma''+1}, \{V_{2,i}'\}_{i=1}^{\sigma''+1}, V_1', V_2'$ from $\mathbb{Z}_N$ and outputs TK_j. Denote $TK_j=\{(u^j, v^j), (\{K_{1,i}^j, K_{2,i}^j\}_{i=1}^{\sigma''+1}, K_1^j, K_2^j)\}_{j=1}^{N_\delta}$, where

$$\left\{\begin{array}{l} \{K_{1,i}^j\}_{i=1}^{\sigma''+1}=\{g_1^{V_{1,i}'}(g_1^{a_1}g_2)^{\beta_1 y_i}(g_3^{d^2})^{T' r_{1,i}^j}\}_{i=1}^{\sigma''+1}=\{g_1^{V_{1,i}} g_2^{\beta_1 y_i} g_3^{T r_{1,i}^j}\}_{i=1}^{\sigma''+1} \\ \{K_{2,i}^j\}_{i=1}^{\sigma''+1}=\{g_1^{V_{2,i}'}(g_1^{a_1}g_2)^{\beta_2, y_i}(g_3^{d})^{T' w_i}\cdot(g_3^{d^2})^{T' r_{2,i}'^j}\}_{i=1}^{\sigma''+1}=\{g_1^{V_{2,i}} g_2^{\beta_2 y_i} g_3^{T r_{2,i}^j}\}_{i=1}^{\sigma''+1} \\ K_1^j=\prod_{i=1}^{\sigma''+1}(K_{1,i}^{-q_{1,i}^j} K_{2,i}^{-q_{2,i}^j})(g_2^{b_1} g_3^{c_1 d})^{V_1'}(g_3^{d})^{V_2'}=g_1^{-\Sigma_{i=1}^{\sigma''+1}(V_{1,i} q_{1,i}^j + V_{2,i} q_{1,i}^j)} g_2^{V_1} g_3^{V_2} \\ K_2^j=(g_3^{d_2})^{T'}=g_3^{T} \end{array}\right\}_{j=1}^{N_\delta}$$

with $(u^j \xleftarrow{R} \mathbb{Z}_{|N_Q|}, v^j \xleftarrow{R} \mathbb{Z}_{|N_Q|}), T=d^2T', \{V_{1,i}=\beta_1 a_1 y_i + V_{1,i}'\}_{i=1}^{\sigma''+1}, \{V_{2,i}=\beta_2 a_1 y_i + V_{2,i}'\}_{i=1}^{\sigma''+1}, \{r_{2,i}^j = d^{-1} w_i + r_{2,i}'^j\}_{i=1}^{\sigma''+1}, V_1 = b_1 V_1' - \beta_1 \Sigma_{i=1}^{\sigma''+1} q_{1,i}^j y_i - \beta_2 \Sigma_{i=1}^{\sigma''+1} q_{2,i}^j y_i$, and $V_2 = c_1 d V_1' + d V_2' - T(\Sigma_{i=1}^{\sigma''+1} q_{1,i}^j) r_{1,i}^j + \Sigma_{i=1}^{\sigma''+1} q_{2,i}^j r_{2,i}^j)$. The tokens generated in Phase 1 has the same distribution as that by **SP-FE.GenToken** because $V_1', V_2' \xleftarrow{R} \mathbb{Z}_N$.

(2) Ciphertext Query. $\mathcal{B}$ receives a plaintext $w = (w_1, \ldots, w_{n_w})$ from $\mathcal{A}$. $\mathcal{B}$ transforms w into $w' = (w_1, \ldots, w_{N_w})$ by **addSpecialSymbols** and transforms w' into $\{\boldsymbol{x}_j\}_{j=1}^{N_w}$ by **symbolSetToVector**. $\mathcal{B}$ randomly sets $S, \alpha_1', \alpha_2', \{U_{1,i}'\}_{i=1}^{\sigma''+1}$, $\{U_{2,i}'\}_{i=1}^{\sigma''+1}, U_1', U_2'$ from $\mathbb{Z}_N$ and outputs CT_j for $\mathcal{A}$. Denote $CT_j=\{\{C_{1,i}^j, C_{2,i}^j\}_{i=1}^{\sigma''+1}$, $C_1^j, C_2^j\}_{j=1}^{N_\delta}$, where

$$\left\{\begin{array}{l}\{C^j_{1,i}\}_{i=1}^{\sigma''+1} = \{(g_1)^{Sq^j_{1,i}}(g_2^{b_1}g_3^{c_1})^{\alpha'_1 x_i}(g_3^{d^2})^{U'_{1,i}}\}_{i=1}^{\sigma''+1}\{g_1^{Sq^j_{1,i}}g_2^{\alpha_1 x_i}g_3^{U_{1,i}}\}_{i=1}^{\sigma''+1}\\ \{C^j_{2,i}\}_{i=1}^{\sigma''+1} = \{(g_1)^{Sq^j_{2,i}}(g_2^{b_1}g_3^{c_1})^{\alpha_2, x_i}(g_3^{d^2})^{U'_{2,i}}\}_{i=1}^{\sigma''+1}\{g_1^{Sq^j_{2,i}}g_2^{\alpha'_2 x_i}g_3^{U_{2,i}}\}_{i=1}^{\sigma''+1}\\ C^j_1 = g_1^S\\ C^j_2 = \prod_{i=1}^{\sigma''+1}[(g_3^{d^2})^{-U'_{1,i}r^j_{1,i}-U'_{2,i}r'^j_{2,i}}(g_3^{c_1})^{\alpha'_1 x_i+\alpha'_1 x_i}].\\ \quad \prod_{i=1}^{\sigma''+1}(g_3^d)^{-U'_{2,i}w_i}\cdot(g_1^{a_1}g_2)^{U'_1}g_1^{U'_2} = g_1^{U_1}g_2^{U_2}g_3^{-\Sigma_{i=1}^{\sigma''+1}(U_{1,i}r^j_{1,i}+U_{2,i}r^j_{2,i})}\end{array}\right\}_{j=1}^{N_\delta}$$

with $\alpha_1{=}b_1\alpha'_1, \alpha_2{=}b_1\alpha'_2, \{U_{1,i}{=}d^2U'_{1,i}{+}c_1\alpha'_1x_i\}_{i=1}^{\sigma''+1}, \{U_{2,i}{=}d^2U'_{2,i}{+}c_1\alpha'_2x_i\}_{i=1}^{\sigma''+1}$, $\{r^j_{2,i}{=}d^{-1}w_i + r'^j_{2,i}\}_{i=1}^{\sigma''+1}, U_1 = a_1U'_1 + U'_2,$ and $U_2 = U'_1$. The ciphertexts generated in Phase 1 has the same distribution as that by **SP-FE.Encrypt** because $U'_1, U'_2 \xleftarrow{R} \mathbb{Z}_N$.

- **Challenge:** $\mathcal{B}$ receives query of challenge token from $\mathcal{A}$. $\mathcal{B}$ is given the challenge query for **Assumption 1** as $T_b = g_1^{a_3}g_2^{\mathrm{b}b_2}g_3^{c_2}$ with $\mathrm{b} \in \{0,1\}$ and $a_3 \xleftarrow{R} \mathbb{Z}_{p_1}$, $b_2 \xleftarrow{R} \mathbb{Z}_{p_2}$, c_2, $d \xleftarrow{R} \mathbb{Z}_{p_3}$. $\mathcal{B}$ randomly sets $\beta_1, \beta_2, \{V'_{1,i}\}_{i=1}^{\sigma''+1}, \{V'_{2,i}\}_{i=1}^{\sigma''+1}, V'_1$, and V'_2 from $\mathbb{Z}_N$ and generates corresponding tokens for $\mathcal{A}$ as follows.

$$\left\{\begin{array}{l}\{K^j_{1,i}\}_{i=1}^{\sigma''+1} = \{g_1^{V'_{1,i}}(g_1^{a_1}g_2)^{\beta_1 w_i}(g_3^{c_2 d})^{r^j_{1,i}}\}_{i=1}^{\sigma''+1} = \{g_1^{V_{1,i}}g_2^{\beta_1 w_i}g_3^{Tr^j_{1,i}}\}_{i=1}^{\sigma''+1}\\ \{K^j_{2,i}\}_{i=1}^{\sigma''+1} = \{(T_{\mathrm{b}})^{w_i}g_1^{V'_{2,i}}(g_3^{c_2 d})^{r'^j_{2,i}}\}_{i=1}^{\sigma''+1} = \{g_1^{V_{2,i}}g_2^{\beta_2 w_i}g_3^{Tr^j_{2,i}}\}_{i=1}^{\sigma''+1}\\ K^j_1 = g_3^{c_2 d} = g_3^T\\ K^j_2 = \Pi_{i=1}^{\sigma''+1}(K_{1,i}^{-q^j_{1,i}}K_{2,i}^{-q^j_{2,i}})(g_3^{c_2 d}g_2^{b_1})^{V'_1}g_3^{V'_2} = g_1^{-\Sigma_{i=1}^{\sigma''+1}(V_{1,i}q^j_{1,i}+V_{2,i}q^j_{1,i})}g_2^{V_1}g_3^{V_2}\end{array}\right\}_{j=1}^{N_\delta}$$

with $(u^j \xleftarrow{R} \mathbb{Z}_{|Q''|}, v^j \xleftarrow{R} \mathbb{Z}_{|Q''|}), T = c_2 d, \beta_2{=}\mathrm{b}b_2, \{V_{1,i}{=}d^2V'_{1,i} + a_1\beta_1 w_i\}_{i=1}^{\sigma''+1}$, $\{V_{2,i} = V'_{2,i} + a_3 w_i\}_{i=1}^{\sigma''+1}, V_1 = b_1V'_1 - \Sigma_{i=1}^{\sigma''+1}(\beta_1 q^j_{1,i} + \beta_2 q^j_{1,i})w_i,$ and $V_2{=}TV'_1 + V'_2 - T\Sigma_{i=1}^{\sigma''+1}(q^j_{1,i}\cdot r^j_{1,i} + q^j_{1,i}r^j_{2,i})$. The distribution of the ciphertexts generated when $T_1 = g_1^{a_3}g_2^{b_2}g_3^{c_2}$ is given is exactly the same as the one in Game 1. Similarly, The distribution of the ciphertexts generated when $T_0 = g_1^{a_3}g_3^{c_2}$ is given is exactly the same as the one in Game 2.

- **Phase 2:** $\mathcal{B}$ continues to adaptively query as in Phase 1.
- **Guess:** $\mathcal{A}$ outputs a guess b' of b and sends it to $\mathcal{B}$.

If the adversary $\mathcal{A}$ has the advantage ϵ in distinguishing Game 1 from Game 2, then the simulator $\mathcal{B}$ has the same ϵ advantage in breaking Assumption 1. This completes the proof of the Lemma 1. □

Lemma 2. *If $\mathcal{G}$ satisfies Assumption* 1*, Game* 2 *and Game* 3 *are computationally indistinguishable.*

Proof: Simulator $\mathcal{B}$ tries to break Assumption 1. by an adversary $\mathcal{A}$ trying to distinguish Game 2 from Game 3. Simulator $\mathcal{B}$ is given an instance of Assumption 1: the public parameter $(N, \mathbb{G}, \mathbb{G}_T, \hat{e})$, $(g_1, g_1^{a_1}, g_2, g_2^{b_1}, g_3^{c_1}, g_3^{c_2 d}, g_3^d, g_3^{d^2}, g_1^{a_2}, g_3^{c_1 d})$ and $a_1, a_2, a_3 \xleftarrow{R} \mathbb{Z}_{p_1}$, $b_1, b_2 \xleftarrow{R} \mathbb{Z}_{p_2}$, $c_1, c_2, d \xleftarrow{R} \mathbb{Z}_{p_3}$, $g_1 \xleftarrow{R} \mathbb{G}_1$, $g_2 \xleftarrow{R} \mathbb{G}_2$ and $g_3 \xleftarrow{R} \mathbb{G}_3$. Also, generate a random bit $\mathrm{b} \in \{0,1\}$, and set $T_b = g_1^{a_3}g_2^{\mathrm{b}b_2}g_3^{c_2}$. -

Setup: $\mathcal{A}$ is given public parameters and outputs the descriptions of two challenges $M_1 = (\Sigma, Q_1, \delta_1 = \{(u^j, v^j, W)\}_{j=1}^{n_\delta}, q_{0,1}, F_1)$ and $M_2 = (\Sigma, Q_2, \delta_2 = \{(u^j, v^j), W\}_{j=1}^{n_\delta}, q_{0,2}, F_2)$. $\mathcal{B}$ converts M_1 into M_1' by **addStatesTransitions** and transforms δ' into $\{(u^j, v^j), (\boldsymbol{w}_j)\}_{j=1}^{N_\delta}$ by **symbolSetToVector**. $\mathcal{B}$ performs the same procedures on M_2 to obtain $\{(u^j, v^j), (\boldsymbol{z}_j)\}_{j=1}^{N_\delta}$. $\mathcal{B}$ sends $\{\boldsymbol{w}_j\}_{j=1}^{N_\delta}$ and $\{\boldsymbol{z}_j\}_{j=1}^{N_\delta}$ to the challenger $\mathcal{C}$ and sets $\{\{q_{1,i}^j, q_{2,i}^j\}_{i=1}^{\sigma''+1}, \{r_{1,i}^j\}_{i=1}^{\sigma''+1}, \{r_{2,i}'^j\}_{i=1}^{\sigma''+1}\}_{j=1}^{N_\delta}$ from $\mathbb{Z}_N$, where $|\Sigma''| = \sigma''$.

- **Phase 1:** $\mathcal{A}$ adaptively outputs one of the two queries.

(1) Token Query. $\mathcal{B}$ receives a predicate M=$(\Sigma, Q, \delta$=$\{(u^j, v^j, W)\}_{j=1}^{n_\delta}, q_0, F)$ from $\mathcal{A}$. $\mathcal{B}$ transforms M into $M'' = (\Sigma'', Q'', \delta'' = \{(u^j, v^j), W\}_{j=1}^{N_\delta}, q_0', F'')$ by **addStatesTransitions** and transforms δ' into $\{(u^j, v^j), (\boldsymbol{y}_j)\}_{j=1}^{N_\delta}$ by **symbolSetToVector**. $\mathcal{B}$ randomly sets $T', \beta_1, \beta_2, \{V_{1,i}'\}_{i=1}^{\sigma''+1}, \{V_{2,i}'\}_{i=1}^{\sigma''+1}, V_1', V_2'$ from $\mathbb{Z}_N$ and outputs TK_j. Denote TK_j=$\{(u^j, v^j), (\{K_{1,i}^j, K_{2,i}^j\}_{i=1}^{\sigma''+1}, K_1^j, K_2^j)\}_{j=1}^{N_\delta}$. The only difference between Lemma 1 and Lemma 2 is that Lemma 2 implicitly sets $\{r_{2,i}^j = d^{-1}z_i + r_{2,i}'\}_{i=1}^{\sigma''+1}$ in $\{K_{2,i}^j\}_{i=1}^{\sigma''+1}$. The tokens in Phase 1 has the same distribution as that by **SP-FE.GenToken** because $V_1', V_2' \xleftarrow{R} \mathbb{Z}_N$.

(2) Ciphertext Query. $\mathcal{B}$ receives a plaintext w=$(w_1, \ldots, w_{n_w})$ from $\mathcal{A}$. $\mathcal{B}$ transforms w into $w' = (w_1, \ldots, w_{N_w})$ by **addSpecialSymbols** and transforms w' into $\{\boldsymbol{x}_j\}_{j=1}^{N_w}$ by **symbolSetToVector**. $\mathcal{B}$ sets $S, \alpha_1', \alpha_2', \{U_{1,i}'\}_{i=1}^{\sigma''+1}, \{U_{2,i}'\}_{i=1}^{\sigma''+1}$, U_1', U_2' from $\mathbb{Z}_N$ and outputs CT_j for $\mathcal{A}$. Denote $CT_j = \{\{C_{1,i}^j, C_{2,i}^j\}_{i=1}^{\sigma''+1}, C_1^j, C_2^j\}_{j=1}^{N_\delta}$. The only difference between Lemma 1 and Lemma 2 is that Lemma 2 implicitly sets $\{r_{2,i}^j = d^{-1}z_i + r_{2,i}'^j\}_{i=1}^{\sigma''+1}$ in $\{K_{2,i}^j\}_{i=1}^{\sigma''+1}$. The ciphertexts generated in Phase 1 has the same distribution as that by **SP-FE.Encrypt** because $U_1', U_2' \xleftarrow{R} \mathbb{Z}_N$. - **Challenge:** $\mathcal{B}$ receives query of challenge token from $\mathcal{A}$. $\mathcal{B}$ is given the challenge query for **Assumption 1** as $T_b = g_1^{a_3} g_2^{\mathrm{b}b_2} g_3^{c_2}$ with b $\in\{0, 1\}$ and $a_3 \xleftarrow{R} \mathbb{Z}_{p_1}$, $b_2 \xleftarrow{R} \mathbb{Z}_{p_2}$, c_2, $d \xleftarrow{R} \mathbb{Z}_{p_3}$. $\mathcal{B}$ randomly generates $\beta_1, \beta_2, \{V_{1,i}'\}_{i=1}^{\sigma''+1}, \{V_{2,i}'\}_{i=1}^{\sigma''+1}, V_1', V_2'$ from $\mathbb{Z}_N$ and generates corresponding tokens for $\mathcal{A}$ as follows.

Denote $TK_j = \{(u^j, v^j), \{K_{1,i}^j, K_{2,i}^j\}_{i=1}^{\sigma''+1}, K_1^j, K_2^j\}_{j=1}^{N_\delta}$, where

$$\left\{\begin{array}{l} \{K_{1,i}^j\}_{i=1}^{\sigma''+1} = \{g_1^{V_{1,i}'}(g_1^{a_1}g_2)^{\beta_1 w_i}(g_3^{c_2 d})^{r_{1,i}^j}\}_{i=1}^{\sigma''+1} = \{g_1^{V_{1,i}} g_2^{\beta_1 w_i} g_3^{Tr_{1,i}^j}\}_{i=1}^{\sigma''+1} \\ \{K_{2,i}^j\}_{i=1}^{\sigma''+1} = \{(T_{\mathrm{b}})^{z_i} g_1^{V_{2,i}'}(g_3^{c_2 d})^{r_{2,i}'^j}\}_{i=1}^{\sigma''+1} = \{g_1^{V_{2,i}} g_2^{\beta_2 z_i} g_3^{Tr_{2,i}^j}\}_{i=1}^{\sigma''+1} \\ K_1^j = g_3^{c_2 d} = g_3^T \\ K_2^j = \Pi_{i=1}^{\sigma''+1}(K_{1,i}^{-q_{1,i}^j} K_{2,i}^{-q_{2,i}^j})(g_3^{c_2 d} g_2^{b_1})^{V_1'} g_3^{V_2'} = g_1^{-\Sigma_{i=1}^{\sigma''+1}(V_{1,i}q_{1,i}^j + V_{2,i}q_{1,i}^j)} g_2^{V_1} g_3^{V_2} \end{array}\right\}_{j=1}^{N_\delta}$$

with $(u^j \xleftarrow{R} \mathbb{Z}_{|Q''|}, v^j \xleftarrow{R} \mathbb{Z}_{|Q''|}), T$=$c_2 d, \beta_2$=$\mathrm{b}b_2, \{V_{1,i}$=$d^2 V_{1,i}' + a_1\beta_1 w_i\}_{i=1}^{\sigma''+1}$, $\{V_{2,i}$=$V_{2,i}' + a_3\beta_2 z_i\}_{i=1}^{\sigma''+1}, V_1 = b_1 V_1' - \Sigma_{i=1}^{\sigma''+1}(\beta_1 q_{1,i}^j w_i + \beta_2 q_{1,i}^j z_i)$, and $V_2 = TV_1' + V_2' - T\Sigma_{i=1}^{\sigma''+1}(q_{1,i}^j \cdot r_{1,i}^j + q_{1,i}^j r_{2,i}^j)$. The distribution of the ciphertexts generated when $T_1 = g_1^{a_3} g_2^{b_2} g_3^{c_2}$ is given is exactly the same as the one in Game 3. Similarly, The distribution of the ciphertexts generated when $T_0 = g_1^{a_3} g_3^{c_2}$ is

given is exactly the same as the one in Game 2.

- **Phase 2:** $\mathcal{B}$ continues to adaptively query as in Phase 1.

- **Guess:** $\mathcal{A}$ outputs a guess b′ of b and sends it to $\mathcal{B}$.

If the adversary $\mathcal{A}$ has the advantage ϵ in distinguishing Game 2 from Game 3, then the simulator $\mathcal{B}$ has the same ϵ advantage in breaking Assumption 1. This completes the proof of the Lemma 2. □

Lemma 3. *If $\mathcal{G}$ satisfies Assumption* 1*, Game* 3 *and Game* 5 *are computationally indistinguishable.*

Proof: Game 3 and Game 4 are computationally indistinguishable following the proof of Lemma 2 by setting Game 4 as Game 2 except for exchanging $\{\{K_{1,i}^j\}_{i=1}^{\sigma''+1}\}_{j=1}^{N_\delta}$ with $\{\{K_{2,i}^j\}_{i=1}^{\sigma''+1}\}_{j=1}^{N_\delta}$ and exchanging $\{(u^j, v^j), \boldsymbol{w}_j\}_{j=1}^{N_\delta}$ with $\{(u^j, v^j), \boldsymbol{z}_j\}_{j=1}^{N_\delta}$. Similarly, Game 4 and Game 5 are computationally indistinguishable following the proof of Lemma 1 by setting Game 4 as Game 2 except for exchanging $\{\{K_{1,i}^j\}_{i=1}^{\sigma''+1}\}_{j=1}^{N_\delta}$ with $\{\{K_{2,i}^j\}_{i=1}^{\sigma''+1}\}_{j=1}^{N_\delta}$ and exchanging $\{(u^j, v^j), \boldsymbol{w}_j\}_{j=1}^{N_\delta}$ with $\{(u^j, v^j), \boldsymbol{z}_j\}_{j=1}^{N_\delta}$. This completes the proof.