Functional Encryption from (Small) Hardware Tokens

Kai-Min Chung[1]*, Jonathan Katz[2]**, and Hong-Sheng Zhou[3]***

[1] Academia Sinica, kmchung@iis.sinica.edu.tw
[2] University of Maryland, jkatz@cs.umd.edu
[3] Virginia Commonwealth University, hszhou@vcu.edu

Abstract. Functional encryption (FE) enables fine-grained access control of encrypted data while promising simplified key management. In the past few years substantial progress has been made on functional encryption and a weaker variant called predicate encryption. Unfortunately, fundamental impossibility results have been demonstrated for constructing FE schemes for general functions satisfying a simulation-based definition of security.

We show how to use *hardware tokens* to overcome these impossibility results. In our envisioned scenario, an authority gives a hardware token and some cryptographic information to each authorized user; the user combines these to decrypt received ciphertexts. Our schemes rely on *stateless* tokens that are *identical* for all users. (Requiring a different token for each user trivializes the problem, and would be a barrier to practical deployment.) The tokens can implement relatively "lightweight" computation relative to the functions supported by the scheme.

Our token-based approach can be extended to support hierarchal functional encryption, function privacy, and more.

1 Introduction

In traditional public-key encryption, a sender encrypts a message M with respect to the public key pk of a particular receiver, and only that receiver (i.e., the owner of the secret key associated with pk) can decrypt the resulting ciphertext and recover the underlying message. More recently, there has been an explosion of interest in encryption schemes that can provide greater flexibility and more refined access to encrypted data. Such schemes allow the sender to specify a *policy* at the time of encryption, and enable any user (decryptor) satisfying the policy (within the given system) to decrypt the resulting ciphertext.

Work in this direction was spurred by constructions of identity-based encryption (IBE) [8], fuzzy IBE [43], and attribute-based encryption [29]. Each of these can be cast as special cases of *predicate encryption* [11, 35], which is in turn a special case of the more powerful notion of *functional encryption* (FE) recently introduced by Boneh, Sahai, and Waters [10]. Roughly speaking, in an FE scheme a user's secret key SK_K is associated with a policy K. Given an encryption of some message M, a user in possession of the secret key SK_K associated with K can recover $F(K, M)$ for some function F fixed as part of the scheme itself. (In the most general case F might be a universal Turing machine, but weaker F are also interesting.)

Security of functional encryption, informally, guarantees that a group of users with secret keys SK_{K_1}, $\ldots$, SK_{K_ℓ} learn nothing from an encryption of M that is not implied by $F(K_1, M)$, $\ldots$, $F(K_\ell, M)$ (plus the length of M). As far as formal definitions are concerned, early work on predicate encryption used an indistinguishability-based definition of security, but Boneh et al. [10] and O'Neill [41] independently showed that such a definitional approach is not, in general, sufficient for analyzing functional encryption. They suggest to use stronger, simulation-based definitions of security (similar in spirit to semantic security) instead.

In the past few years substantial progress has been made in this area [42, 25, 4, 26, 17, 24, 18, 3, 14, 1]. Yet several open questions remain. First, it remains an unsolved problem to construct an FE scheme for *arbitrary* functions F with *unbounded* collusion resistance. Second, it is unknown how to realize the strongest simulation-based notion of security for functional encryption. In fact, Boneh et al. [10] and Agrawal et al. [1] showed fundamental limitations on achieving such definitions for FE schemes supporting arbitrary F.

* Work done while at Cornell University and supported by NSF awards CNS-1217821 and CCF-1214844, and the Sloan Research Fellowship of Rafael Pass.

** Work supported by NSF award #1223623.

*** Work done while at the University of Maryland and supported by an NSF CI postdoctoral fellowship.

Here we propose the use of (stateless) *hardware tokens* to solve both the above issues. In our envisioned usage scenario, an authority gives a hardware token along with a cryptographic key SK_K to each authorized user; the user combines these in order to decrypt received ciphertexts. We believe this would be a feasible approach for realizing functional encryption in small- or medium-size organizations where an authority could purchase hardware tokens, customize them as needed, and then give them directly to users in the system.

The idea of using physical devices to bypass cryptographic impossibility results has been investigated previously. Katz [34] showed that hardware tokens can be used for universally composable computation of arbitrary functions. His work motivated an extensive amount of follow-up work [12, 40, 13, 27, 28, 15]. In the context of program obfuscation, several works [28, 6, 16] considered using hardware tokens to achieve program obfuscation, which is impossible in the plain model even for simple classes of programs [2].

A token-based approach should have the following properties:

1. The tokens used should be *universal*, in the sense that every user in the system is given an *identical* token. Having a single token used by everyone appears to be the only way to make a token-based approach viable.
2. In applications where the complexity of F is high, it is desirable that tokens be "lightweight" in the sense that the complexity of the token is smaller than the complexity of F.

In this work, we show token-based solutions that satisfy the above requirements. Additionally, our constructions satisfy a strong simulation-based notion of security and have succinct ciphertexts (of size independent of F). We provide the intuition behind our approach in the next section.

1.1 Our Results

Let pk, sk denote the public and secret keys for a (standard) public-key encryption scheme. Intuitively, a trivial construction of an FE scheme based on (stateless) tokens is to let pk be the master public key, and to give the user associated with key K a token which implements the functionality $\mathsf{token}_{sk,K}(C) = F(K, \mathsf{Dec}_{sk}(C))$. In this scheme the tokens are not *universal*, as each user must be given a token whose functionality depends on that user's key K. Perhaps more surprising is that this scheme is not secure: nothing prevents a user from modifying C before feeding the ciphertext to its token; if the encryption scheme scheme is *malleable* then a user might be able to use such a cheating strategy to learn disallowed information about the underlying message M. We will address both these issues in our solutions, described next.

Solution #1. We can address the universality issue by having the user provide K along with C as input to the token. (The token will then implement the functionality $\mathsf{token}_{sk}(K, C) = F(K, \mathsf{Dec}_{sk}(C))$.) Now we must prevent the user from changing either K or C. Modifications of the key are handled by *signing* K and hard-coding the verification key vk into the token; the token then verifies a signature on K before decrypting as before. We show that illegal modification of the ciphertext can be solved if the public-key encryption scheme is CCA2-secure; we give the details in Section 4.2.

Solution #2. In the previous solution, the complexity of the token was (at least) the complexity of computing F itself. We can use ideas from the area of verifiable outsource of computation [19] in order to obtain a solution in which the complexity of the token is independent of the complexity of F. The basic idea here is for the token to "outsource" most of the computation of F to the user. To do so, we now let the underlying public-key encryption scheme be *fully homomorphic* [21]. Given a ciphertext $C = \mathsf{Enc}_{pk}(M)$, the user can now compute the transformed ciphertext $\hat{C} = \mathsf{Enc}_{pk}(F(K, M))$ and feed this to the token for decryption. To enforce correct behavior with lightweight tokens, we let the user provide a *succinct non-interactive argument (SNARG)* [23, 20, 5] that the computation is done correctly.[4] The immediate problem with this approach is that any fully-homomorphic encryption scheme is completely malleable! We here instead rely on simulation-extractable non-interactive zero-knowledge proofs (NIZKs) to deal with the malleable issue, where we let the encryptor provide an NIZK proof that $C = \mathsf{Enc}_{pk}(M)$ is correctly encrypted. The computation done by the token now involves (1) verifying the signature on K as in previous solution, and (2) verifying the given

[4] As a technical note, while SNARGs are constructed based on knowledge-type assumptions, we here rely on SNARGs for $\mathbf{P}$, which can be formulated as a falsifiable assumption.

SNARG and NIZK, (3) decrypting the given ciphertext, all of which have complexity independent of F. We give the details in Section 4.3.

While both of our schemes are simple, we show in Section 6 that both schemes satisfy a very strong notion of simulation-based security, where an adversary A gets full access to the scheme (in particular, A can make an arbitrary number of key queries and encryption queries in a fully adaptive way), yet cannot learn any information beyond what it should have learnt through the access. At a high level, our security proof crucially relies on the fact that in the simulation, the simulator gets to simulate token's answers to the queries made by the adversary, which bypasses the information-theoretic arguments underlying the impossibility results of Boneh et al. [10] and Agrawal et al. [1].

We remark that our constructions and the way we get around impossibility results share some similarities to the work of Bitansky et al [6] on achieving program obfuscation using stateless and universal hardware tokens, but the security notion of functional encryption and program obfuscation are different and the results from both contexts does not seem to imply each other. For example, one may think intuitively that by obfuscating the decryption circuit $\mathsf{Dec}_{sk,K}(C) = F(K, \mathsf{Dec}_{sk}(C))$, one obtains a "trivial" construction of functional encryption. However, such a construction cannot satisfy simulation-based security as it does not bypass the impossibility results of [10, 1].

1.2 Extensions

Our approach can be extended in several ways; we sketch two extensions here.

Hierarchical functional encryption. Consider an encrypted database in a company where the top-level manager has the access control on the database that allows different first-level departments to access different part of the data; then any first level department, say, research department, allows different second level subdepartment to run different analytic/learning algorithms over the encrypted data; this naturally induces a hierarchical access structure to the data. To support this natural variant of access control, we need *hierarchical functional encryption*, which generalizes many primitives considered in the literature, such as hierarchical IBE [33, 22, 7, 44, 37], hierarchical PE [36].

More precisely, to enable such a hierarchical structure, the global authority may delegate a first level user Alice (under some functionality key K_{Alice} with respect to functionality F_1) the ability to generate a second level secret key $\mathsf{SK}_{K_{Alice}:K_{Bob}}$ of functionality key K_{Bob} with respect to functionality F_2 for a second level user Bob. For a message M encrypted under global master public key, Bob should be able to decrypt $F_2(K_{Bob}, F_1(K_{Alice}, M))$ using $\mathsf{SK}_{K_{Alice}:K_{Bob}}$. Alice may further delegate Bob the ability to generate a third level secret key $\mathsf{SK}_{K_{Alice}:K_{Bob}:K_{Carol}}$ of functionality key K_{Carol} with respect to functionality F_3, and so on.

Our solution #1 can be readily extended to the hierarchical setting using the idea of signature chains. Roughly, to delegate Alice such power, the global authority generates a key pair $(\mathsf{sk}_{Alice}, \mathsf{vk}_{Alice})$ of a digital signature scheme and "authorizes" it by signing $(K_{Alice}, \mathsf{vk}_{Alice})$; sk_{Alice} is given to Alice as a "delegation key" and $(K_{Alice}, \mathsf{vk}_{Alice})$ together with its signature are published. Alice can then generate $\mathsf{SK}_{K_{Alice}:K_{Bob}}$ by simply signing $K_{Alice} : K_{Bob}$ (using sk_{Alice}). To decrypt, Bob queries the token with the ciphertext together with the chain of signatures—including $(K_{Alice}, \mathsf{vk}_{Alice})$ together with its signature and $K_{Alice} : K_{Bob}$ together with its signature. The token returns $F_2(K_{Bob}, F_1(K_{Alice}, M))$ if the chain verifies. Alice can perform further delegation in a similar fashion.

The above solution has the drawback that all functionalities in the hierarchy need to be determined in the global setup and hard-wired in the token. We can further allow adaptively chosen functionalities by further "authorizing" the functionality as well, but at the price that the token needs to receive a description of the functionality (together with its authorization info) as its input, which results in long query length. This issue can be addressed in the framework of our solution #2, where the token only requires a succinct authorization information of the functionality (as such, the complexity of the token remains independent of the functionalities). We provide further details in the full version of this paper.

Function privacy. In a general FE scheme the secret key SK_K may leak K. Preventing such leakage is given as an interesting research direction in [10]. Very recently, Boneh et al. [9] studied the notion of *function privacy* for IBE, and gave several constructions. We can modify our token-based constructions to obtain function privacy in functional encryption: in the key generation, instead of obtaining a signature of K, the

users obtain an encrypted version signature $\mathcal{E}(\sigma)$; the decryption key $\mathtt{sk}$ will be stored in token; at any moment when the users receive a ciphertext C, instead of providing (C, K, σ), the tuple $(C, K, \mathcal{E}(\sigma))$ will be given to token; the token would first decrypt $\mathcal{E}(\sigma)$ into σ and then verify that σ is a valid signature on K and, if so, return the result $F(K, \mathsf{Dec}_{sk}(C))$ as in basic functional encryption constructions.

2 Preliminaries

2.1 Fully Homomorphic Encryption

A fully homomorphic encryption scheme $\mathcal{FHE} = (\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ is a public-key encryption scheme that associates with an additional polynomial-time algorithm Eval, which takes as input a public key ek, a ciphertext $\mathsf{ct} = \mathsf{Enc}_{\mathsf{ek}}(m)$ and a circuit C, and outputs, a new ciphertext $\mathsf{ct}' = \mathsf{Eval}_{\mathsf{ek}}(\mathsf{ct}, C)$, such that $\mathsf{Dec}_{\mathsf{dk}}(\mathsf{ct}') = C(m)$, where dk is the secret key corresponding to the public key ek. It is required that the size of $\mathsf{ct}' = \mathsf{Eval}_{\mathsf{ek}}(\mathsf{Enc}_{\mathsf{ek}}(m), C)$ depends polynomially on the security parameter and the length of $C(m)$, but is otherwise independent of the size of the circuit C. We also require that Eval is deterministic, and the scheme has perfect correctness (i.e. it always holds that $\mathsf{Dec}_{\mathsf{dk}}(\mathsf{Enc}_{\mathsf{ek}}(m)) = m$ and that $\mathsf{Dec}_{\mathsf{dk}}(\mathsf{FHE.Eval}_{\mathsf{ek}}(\mathsf{Enc}_{\mathsf{ek}}(m), C)) = C(m)$). Most known schemes satisfies these properties. For security, we simply require that $\mathcal{FHE}$ is semantically secure.

Since the breakthrough of Gentry [21], several fully homomorphic encryption schemes have been constructed with improved efficiency and based on more standard assumptions such as LWE (Learning With Error). In general, these constructions achieve leveled FHE, where the complexity of the schemes depend linearly on the depth of the circuits C that are allowed as inputs to Eval. However, under the additional assumption that these constructions are circular secure (i.e., remain secure even given an encryption of the secret key), the complexity of the schemes are independent of the allowed circuits, and the schemes can evaluate any polynomial-sized circuit.

2.2 Non-interactive Zero-Knowledge Arguments

Let R be a binary relation that is efficiently computable. Let L_R be the language defined by the relation R, that is, $L_R = \{x : \exists w \text{ s.t.} (x, w) \in R\}$. For any pair $(x, w) \in R$, we call x the statement and w the witness.

Definition 1 (NIZK). *A tuple of* PPT *algorithms* $\mathcal{NIZK} = (\mathsf{NIZK.Gen}, \mathsf{NIZK}.P, \mathsf{NIZK}.V)$*, is a non-interactive zero-knowledge (NIZK) argument system for* R *if it has the following properties described below:*

Completeness. *For any* $(x, w) \in R$ *it holds that*

$$\Pr[\mathsf{crs} \leftarrow \mathsf{NIZK.Gen}(1^\kappa); \pi \leftarrow \mathsf{NIZK}.P(\mathsf{crs}, x, w) : \ \mathsf{NIZK}.V(\mathsf{crs}, x, \pi) = 1] = 1.$$

Soundness. *For any non-uniform* PPT $\mathcal{A}$*, it holds that*

$$\Pr[\mathsf{crs} \leftarrow \mathsf{NIZK.Gen}(1^\kappa); (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) : \ \mathsf{NIZK}.V(\mathsf{crs}, x, \pi) = 1] \leq \mathrm{negl}(\kappa).$$

Zero-knowledge. *For any non-uniform* PPT $\mathcal{A}$*, there exists a* PPT $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *such that it holds that* $|p_1 - p_2| \leq \mathrm{negl}(\kappa)$*, where*

$$p_1 = \Pr[\mathsf{crs} \leftarrow \mathsf{NIZK.Gen}(1^\kappa) : \mathcal{A}^{\mathsf{NIZK}.P(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) = 1]$$
$$p_2 = \Pr[(\mathsf{crs}, \tau, \xi) \leftarrow \mathcal{S}_1(1^\kappa) : \mathcal{A}^{Sim(\mathsf{crs},\tau,\cdot,\cdot)}(\mathsf{crs}) = 1]$$

where $Sim(\mathsf{crs}, \tau, x, w) = \mathcal{S}_2(\mathsf{crs}, \tau, x)$ *for* $(x, w) \in R$*. Both oracles* $\mathsf{NIZK}.P()$ *and* $Sim()$ *output* $\perp$ *if* $(x, w) \notin R$*.*

Next we define (unbounded) simulation-extractability of NIZK [30, 32]. Intuitively, it says that even after seeing many simulated proofs, whenever the adversary makes a new proof we are able to extract a witness.

Definition 2 (Simulation-Extractability). *Let* $\mathcal{NIZK} = (\mathsf{NIZK.Gen}, \mathsf{NIZK}.P, \mathsf{NIZK}.V)$ *be a NIZK argument system for* R*. We say* $\mathcal{NIZK}$ *is* simulation-extractable *if for all* PPT *adversaries* $\mathcal{A}$*, there exists a* PPT $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ *so that*

$$\Pr\left[\begin{array}{c}(\mathsf{crs}, \tau, \xi) \leftarrow \mathcal{S}_1(1^\kappa); (x, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_2(\mathsf{crs},\tau,\cdot)}(\mathsf{crs}); w \leftarrow \mathcal{S}_3(\mathsf{crs}, \xi, x, \pi) : \\ \mathsf{NIZK}.V(\mathsf{crs}, x, \pi) = 1 \ \wedge \ (x, \pi) \notin Q \ \wedge \ (x, w) \notin R\end{array}\right] \leq \mathrm{negl}(\kappa)$$

where Q *is the list of simulation queries and responses* (x_i, π_i) *that* $\mathcal{A}$ *makes to* $\mathcal{S}_2()$*.*

2.3 SNARG

We present the definition of succinct non-interactive arguments (abbreviated SNARGs) [23, 20, 5]. A SNARG for a function class $\mathcal{F} = \{\mathcal{F}_\kappa\}_{\kappa\in\mathbb{N}}$ consists of a set of PPT algorithms $\mathcal{SNARG} = \mathsf{SNARG}.\{\mathsf{Gen}, P, V\}$: The generation algorithm Gen on input security parameter 1^κ and a function $F : \{0,1\}^{n(\kappa)} \to \{0,1\}^{m(\kappa)} \in \mathcal{F}_\kappa$ (represented as a circuit), outputs a reference string rs and a (short) verification state vrs.[5] The prover P on input rs and an input string $x \in \{0,1\}^n$, outputs an answer $y = F(x)$ together with a (short) proof ϖ. The verifier algorithm V on input vrs, x, y, and ϖ outputs a bit $b \in \{0,1\}$ represents whether V accepts or rejects. We require the following properties for a SNARG scheme.

- **Completeness:** For every $\kappa \in \mathbb{N}$, $F \in \mathcal{F}_\kappa$, $x \in \{0,1\}^n$, the probability that the verifier V rejects in the following experiment is negligible in κ: (i) $(\mathsf{rs}, \mathsf{vrs}) \leftarrow \mathsf{Gen}(1^\kappa, F)$, (ii) $(y, \varpi) \leftarrow P(\mathsf{rs}, x)$, and (iii) $b \leftarrow V(\mathsf{vrs}, x, y, \varpi)$.
- **Soundness:** For every efficient adversary P^*, and every $\kappa \in \mathbb{N}$, the probability that P^* makes V accept an incorrect answer in the following experiment is negligible in κ: (i) P^* on input 1^κ outputs a function $F \in \mathcal{F}_\kappa$, (ii) $(\mathsf{rs}, \mathsf{vrs}) \leftarrow \mathsf{Gen}(1^\kappa, F)$, (iii) P^* on input rs, outputs x, y, ϖ with $y \neq F(x)$, and (iv) $b \leftarrow V(\mathsf{vrs}, x, y, \varpi)$.
- **Efficiency:** The running time of the verifier is $\mathrm{poly}(\kappa, n + m, \log |F|)$ (which implies the succinctness of vrs and ϖ). The running time of the generation algorithm and the prover is $\mathrm{poly}(\kappa, |F|)$.

We say $\mathcal{SNARG}$ is *publicly-verifiable* if the verification state vrs is part of the reference string rs.

We require a publicly-verifiable SNARG scheme $\mathcal{SNARG}$ for polynomial-size circuits. Such a SNARG scheme can be obtained by using Micali's CS proof [39] (with random oracle instantiated by some hash function heuristically), or provably secure based on publicly-verifiable succinct non-interactive arguments (SNARGs), which in turn can be constructed based on (non-falsifiable) q-PKE (q-power knowledge of exponent) and q-PDH (q-power Diffie-Hellman) assumptions on bilinear groups. Such SNARGs was first constructed implicitly in [31] and later improved by [38, 20], where [20] explicitly constructs SNARGs. In the scheme of [20], the generation algorithm and the prover run in time quasi-linear in the size of F with rs length linear in $|F|$, and the verifier runs in linear time in the input and output length.

3 Definition of Functional Encryption

Functional encryption was recently introduced by Boneh, Sahai, and Waters [10]. Let $\mathcal{F} = \{\mathcal{F}_\kappa\}_{\kappa\in\mathbb{N}}$ where $\mathcal{F}_\kappa = \{F : \mathcal{K}_\kappa \times \mathcal{M}_\kappa \to \mathcal{M}_\kappa\}$ be an ensemble of functionality class indexed by a security parameter κ. A functional encryption scheme $\mathcal{FE}$ for a functionality class $\mathcal{F}$ consists of four PPT algorithms $\mathcal{FE} = \mathsf{FE}.\{\mathsf{Setup}, \mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}\}$ defined as follows.

- **Setup:** $\mathsf{FE.Setup}(1^\kappa, F)$ is a PPT algorithm takes as input a security parameter 1^κ and a functionality $F \in \mathcal{F}_\kappa$ and outputs a pair of master public and secret keys $(\mathsf{MPK}, \mathsf{MSK})$.
- **Key Generation:** $\mathsf{FE.Key}(\mathsf{MSK}, K)$ is a PPT algorithm that takes as input the master secret key MSK and a functionality key $K \in \mathcal{K}_\kappa$ and outputs a corresponding secret key SK_K.
- **Encryption:** $\mathsf{FE.Enc}(\mathsf{MPK}, M)$ is a PPT algorithm that takes as input the master public key MPK and a message $M \in \mathcal{M}_\kappa$ and outputs a ciphertext CT.
- **Decryption:** $\mathsf{FE.Dec}(\mathsf{SK}_K, \mathsf{CT})$ is a deterministic algorithm that takes as input the secret key SK_K and a ciphertext $\mathsf{CT} = \mathsf{Enc}(\mathsf{MPK}, M)$ and outputs $F(K, M)$.

Definition 3 (Correctness). *A functional encryption scheme $\mathcal{FE}$ is correct if for every $\kappa \in \mathbb{N}$, $F \in \mathcal{F}_\kappa$, $K \in \mathcal{K}_\kappa$, and $M \in \mathcal{M}_\kappa$,*

$$\Pr\left[\begin{array}{l}(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^\kappa, F);\\ \mathsf{FE.Dec}(\mathsf{FE.Key}(\mathsf{MSK}, K), \mathsf{FE.Enc}(\mathsf{MPK}, M)) \neq F(K, M)\end{array}\right] = \mathrm{negl}(\kappa)$$

where the probability is taken over the coins of $\mathsf{FE.Setup}$, $\mathsf{FE.Key}$, *and* $\mathsf{FE.Enc}$.

[5] We assume w.l.o.g. that rs contains a description of F.

We next define a stronger simulation-based notion of security for functional encryption than the existing simulation-based security notions in the literature. We note that, while there are negative results [10, 1] showing that even significantly weaker notions of security are impossible to achieve in the plain model, our token-based construction in Section 4 achieves our strong security notion in the token model.

Our definition is stronger in the sense that we allow the adversary $\mathcal{A}$ to take *full* control over the access of the encryption scheme, where $\mathcal{A}$ can choose the functionality F and request to see an arbitrary number of secret keys SK_K's and ciphtertexts CT's in a *fully adaptive* fashion. Previous definitions either restrict the number of ciphertext queries and/or restrict the order of secret key and ciphertext queries (e.g., require $\mathcal{A}$ to ask for all challenge ciphertexts at once). Informally, the following definition says that even with full access to the encryption scheme, $\mathcal{A}$ still cannot learn any additional information than what it should have legally learnt from the received ciphertexts (using the received secret keys). This, as usual, is formalized by requiring that the ciphertexts can be simulated by an efficient simulator with only the "legal" information.

Definition 4 (Fully-Adaptive Simulation Security). *Let $\mathcal{FE}$ be a functional encryption scheme for a functionality class $\mathcal{F}$. For every* PPT *stateful adversary $\mathcal{A}$ and* PPT *stateful simulator* Sim*, consider the following two experiments.*

$\mathbf{Expt}^{real}_{\mathcal{FE},\mathcal{A}}(1^\kappa)$

1: $F \leftarrow \mathcal{A}(1^\kappa)$;
2: $(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^\kappa, F)$;
PROVIDE MPK TO $\mathcal{A}$;
3: LET $i := 1$;
4: DO
$M_i \leftarrow \mathcal{A}^{\mathsf{FE.Key}(\mathsf{MSK},\cdot)}()$;
$\mathsf{CT}_i \leftarrow \mathsf{FE.Enc}(\mathsf{MPK}, M_i)$;
PROVIDE CT_i TO $\mathcal{A}$;
$i := i + 1$;
UNTIL $\mathcal{A}$ BREAKS;
5: $\alpha \leftarrow \mathcal{A}()$;
6: OUTPUT $(\alpha, \{M_j\}_{j\in[i-1]})$;

$\mathbf{Expt}^{ideal}_{\mathcal{FE},\mathcal{A},\mathsf{Sim}}(1^\kappa)$

1: $F \leftarrow \mathcal{A}(1^\kappa)$;
2: $\mathsf{MPK} \leftarrow \mathsf{Sim}(1^\kappa, F)$;
PROVIDE MPK TO $\mathcal{A}$;
3: LET $i := 1$;
4: DO
$M_i \leftarrow \mathcal{A}^{\mathsf{Sim}^{\{F(\cdot,M_j)\}_{j<i}}}()$;
$\mathsf{CT}_i \leftarrow \mathsf{Sim}^{\{F(\cdot,M_j)\}_{j\le i}}(1^{|M_i|})$;
PROVIDE CT_i TO $\mathcal{A}$;
$i := i + 1$;
UNTIL $\mathcal{A}$ BREAKS;
5: $\alpha \leftarrow \mathcal{A}()$;
6: OUTPUT $(\alpha, \{M_j\}_{j\in[i-1]})$;

In Step 4 of the ideal experiment, Sim *needs to provide answers to* $\mathsf{Key}(\mathsf{MSK},\cdot)$ *queries of $\mathcal{A}$. During the execution of the ideal experiment, we say that* Sim*'s query K to oracles $\{F(\cdot, M_1), \ldots, F(\cdot, M_i)\}$ is* legal *if $\mathcal{A}$ already requested ciphertexts for $M_1, \ldots, M_i$, and made oracle query K to* $\mathsf{Key}(\mathsf{MSK},\cdot)$*. We call a simulator algorithm* Sim admissible *if it only makes legal queries to its oracle throughout the execution.*

The functional encryption scheme $\mathcal{FE}$ is said to be fully-adaptive simulation-secure *if there is an admissible* PPT *stateful simulator* Sim *such that for every* PPT *stateful adversary $\mathcal{A}$, the following two distributions are computationally indistinguishable:*

$$\left\{\mathbf{Expt}^{real}_{\mathcal{FE},\mathcal{A}}(1^\kappa)\right\}_\kappa \overset{c}{\approx} \left\{\mathbf{Expt}^{ideal}_{\mathcal{FE},\mathcal{A},\mathsf{Sim}}(1^\kappa)\right\}_\kappa$$

4 Token Model and Constructions

4.1 Token-based FE

Here we introduce a simple token model for encryption schemes and provide formal definitions of token-based functional encryption schemes. In our model, we consider *stateless* tokens that are initialized by the master authority in the setup stage, and are only used by users in decryption. Furthermore, we require token to be *universal* in the sense that tokens used by different users are identical. Thus, tokens are simply deterministic oracles that are generated by the Setup algorithm, and queried by the Dec algorithm.

Definition 5 (Token-based FE). *A token-based functional encryption scheme $\mathcal{FE}$ is defined identical to the definition of functional encryption scheme except for the following modifications.*

– **Setup:** *In addition to* MPK *and* MSK*, the algorithm* FE.Setup *also outputs a token* $\mathbf{T}$*, which is simply a deterministic oracle.*
– **Key Generation :** *In addition to the keys* SK_K*, the algorithm* FE.Key *also returns a copy of the token* $\mathbf{T}$ *to users.*
– **Decryption:** *The decryption algorithm* $\mathsf{FE.Dec}^{\mathbf{T}}$ *can query the* $\mathbf{T}$ *in order to decrypt.*

The correctness property extends straightforwardly. For security, we generalize fully-adaptive simulation security to the token model. As before, we allow the adversary $\mathcal{A}$ to take full control over the access of the encryption scheme; in particular, $\mathcal{A}$ is given the oracle access to token after setup. In the ideal world, the simulator is required to simulate answers to all queries made by $\mathcal{A}$, including the token queries, given only the "legal" information that $\mathcal{A}$ can learn from the received ciphertexts using the received secret keys.

Definition 6 (Fully-Adaptive Simulation Security for Token-Based FE). *Let* $\mathcal{FE}$ *be a token-basd functional encryption scheme for a functionality class* $\mathcal{F}$*. For every* PPT *stateful adversary* $\mathcal{A}$ *and* PPT *stateful simulator* Sim*, consider the following two experiments.*

$\mathbf{Expt}^{real}_{\mathcal{FE},\mathcal{A}}(1^\kappa)$

1: $F \leftarrow \mathcal{A}(1^\kappa)$;
2: $(\mathsf{MPK}, \mathsf{MSK}, \mathbf{T}) \leftarrow \mathsf{FE.Setup}(1^\kappa, F)$;
  PROVIDE MPK TO $\mathcal{A}$;
3: LET $i := 1$;
4: DO
    $M_i \leftarrow \mathcal{A}^{\mathsf{FE.Key}(\mathsf{MSK},\cdot),\mathbf{T}(\cdot)}()$;
    $\mathsf{CT}_i \leftarrow \mathsf{FE.Enc}(\mathsf{MPK}, M_i)$;
    PROVIDE CT_i TO $\mathcal{A}$;
    $i := i + 1$;
  UNTIL $\mathcal{A}$ BREAKS;
5: $\alpha \leftarrow \mathcal{A}()$;
6: OUTPUT $(\alpha, \{M_j\}_{j\in[i-1]})$;

$\mathbf{Expt}^{ideal}_{\mathcal{FE},\mathcal{A},\mathsf{Sim}}(1^\kappa)$

1: $F \leftarrow \mathcal{A}(1^\kappa)$;
2: $\mathsf{MPK} \leftarrow \mathsf{Sim}(1^\kappa, F)$;
  PROVIDE MPK TO $\mathcal{A}$;
3: LET $i := 1$;
4: DO
    $M_i \leftarrow \mathcal{A}^{\mathsf{Sim}^{\{F(\cdot,M_j)\}_{j<i}}}()$;
    $\mathsf{CT}_i \leftarrow \mathsf{Sim}^{\{F(\cdot,M_j)\}_{j\le i}}(1^{|M_i|})$;
    PROVIDE CT_i TO $\mathcal{A}$;
    $i := i + 1$;
  UNTIL $\mathcal{A}$ BREAKS;
5: $\alpha \leftarrow \mathcal{A}()$;
6: OUTPUT $(\alpha, \{M_j\}_{j\in[i-1]})$;

In Step 4 of the ideal experiment, Sim *needs to provide answers to both* $\mathsf{Key}(\mathsf{MSK}, \cdot)$ *and* $\mathbf{T}(\cdot)$ *queries of* $\mathcal{A}$*. During the execution of the ideal experiment, we say that* Sim*'s query* K *to oracles* $\{F(\cdot, M_1), \ldots, F(\cdot, M_i)\}$ *is* legal *if* $\mathcal{A}$ *already requested ciphertexts for* $M_1, \ldots, M_i$*, and made oracle query* K *to* $\mathsf{Key}(\mathsf{MSK}, \cdot)$*. We call a simulator algorithm* Sim admissible *if it only makes legal queries to its oracle throughout the execution.*

The functional encryption scheme $\mathcal{FE}$ *is said to be* fully-adaptive simulation-secure *if there is an admissible* PPT *stateful simulator* Sim *such that for every* PPT *stateful adversary* $\mathcal{A}$*, the following two distributions are computationally indistinguishable:*

$$\left\{\mathbf{Expt}^{real}_{\mathcal{FE},\mathcal{A}}(1^\kappa)\right\}_\kappa \overset{c}{\approx} \left\{\mathbf{Expt}^{ideal}_{\mathcal{FE},\mathcal{A},\mathsf{Sim}}(1^\kappa)\right\}_\kappa$$

4.2 Token-based FE Construction — Solution #1

Here we give the construction of a functional encryption scheme $\mathcal{FE} = \mathsf{FE.}\{\mathsf{Setup}, \mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}\}$ for a functionality F based on stateless and universal tokens. Our construction is based on a CCA2-secure public key encryption $\mathsf{PKE.}\{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}\}$ and a strongly unforgeable signature scheme $\mathsf{SIG.}\{\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy}\}$. In the setup stage, the authority generates a key-pair $(\mathsf{ek}, \mathsf{dk})$ for encryption and a key-pair $(\mathsf{vk}, \mathsf{sk})$ for digital signature, and set $\mathsf{MPK} = (\mathsf{ek}, \mathsf{vk})$ and $\mathsf{MSK} = \mathsf{sk}$. Additionally, the authority initializes the token $\mathbf{T}$ with the description of F, public keys ek, vk, and secret decryption key dk.

To encrypt a message M, one simply encrypts it using the underlying CCA2 public key ek; that is, the ciphertext is $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}}(M)$. The secret key SK_K for a functionality key K is simply a signature of K; that is, $\mathsf{SK}_K = \sigma_K \leftarrow \mathsf{SIG.Sign}_{\mathsf{sk}}(K)$. To decrypt ct using secret key SK_K, the user queries its token $\mathbf{T}$ with $(\mathsf{ct}, K, \sigma_K)$. $\mathbf{T}$ verifies if σ_K is valid, and if so, $\mathbf{T}$ returns $F(K, \mathsf{PKE.Dec}_{\mathsf{dk}}(\mathsf{ct}))$, and returns $\perp$ otherwise. A formal description of our scheme can be found in Figure 1.

- **Setup:** on input a security parameter 1^κ, a functionality $F \in \mathcal{F}_\kappa$, the setup algorithm $\mathsf{Setup}()$ performs the following steps to generate MPK, MSK, and a deterministic stateless token $\mathbf{T}$.
  - Execute $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PKE.Gen}(1^\kappa)$, and $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Gen}(1^\kappa)$.
  - Initiate a token $\mathbf{T}$ with values $(\mathsf{dk}, \mathsf{ek}, \mathsf{vk}, F)$.
  - Output $\mathsf{MPK} = (\mathsf{ek}, \mathsf{vk})$, and $\mathsf{MSK} = (\mathsf{sk})$.
- **Key Generation:** on input a master secret key MSK and a functionality key K, the key generation algorithm $\mathsf{Key}()$ generates SK_K as follows.
  - Execute $\sigma_K \leftarrow \mathsf{SIG.Sign}_{\mathsf{sk}}(K)$. Output $\mathsf{SK}_K = (\sigma_K)$.
- **Encryption:** on input a master public key MPK and a message M, the encryption algorithm $\mathsf{Enc}()$ generates CT as follows.
  - Execute $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}}(M; \rho)$, where ρ is the randomness. Return $\mathsf{CT} = (\mathsf{ct})$.
- **Decryption:** on input $\mathsf{SK}_K = (\sigma_K)$ and a ciphertext $\mathsf{CT} = (\mathsf{ct})$ of a message M, with access to a token $\mathbf{T}$, the decryption algorithm $\mathsf{Dec}^{\mathbf{T}}()$ performs the following steps to decrypt $m = F(K, M)$:
  - Query the token $m \leftarrow \mathbf{T}(\mathsf{CT}, K, \mathsf{SK}_K)$. Output m.
- **Token Operations:** on query $(\mathsf{CT}, K, \mathsf{SK}_K)$, where $\mathsf{CT} = (\mathsf{ct})$ and $\mathsf{SK}_K = (\sigma_K)$, the token $\mathbf{T}$ carries out the following operations.
  - Execute $\mathsf{SIG.Vrfy}_{\mathsf{vk}}(K, \sigma_K)$.
  - If the above verification accepts, then compute $M \leftarrow \mathsf{PKE.Dec}_{\mathsf{dk}}(\mathsf{ct})$ and return $m = F(K, M)$. Otherwise, return $\perp$.

Fig. 1. Solution #1. Here $\mathsf{PKE.}\{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}\}$ is a public-key encryption scheme, and $\mathsf{SIG.}\{\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy}\}$ is a signature scheme.

Note that our scheme has succinct ciphertext size. Indeed, our ciphertext is simply a CCA2 encryption of the message, which is independent of the complexity of F. On the other hand, our token need to evaluate F to decrypt. Thus, our solution #1 is suitable for lower complexity functionalities (e.g., inner product functionality).

While our scheme is very simple, it satisfies the strong fully-adaptive simulation-security as defined in Definition 6. In fact, the security proof is rather straightforward: The simulator simply simulates Setup and Key queries honestly, and simulates encryption queries M_i by encryption of $0^{|M_i|}$. To answer a token query $(\mathsf{ct}, K, \sigma_K)$, when σ_K verifies, the simulator checks if ct is one of the simulated ciphertext (for some encryption query M_i). If so, the simulator queries its oracle and returns $F(K, M_i)$, and if not, it simulates the token honestly. Intuitively, the simulation works since by strong unforgeability, the simulator can learn correct answers for simulated ciphertexts from its oracle, and CCA2-security ensures that the simulation works for other ciphertexts.

We note that our security proof crucially relies on the fact that in the simulation, the simulator gets to simulate token's answers to the queries made by the adversary, which bypasses the information-theoretic arguments underlying the impossibility results of Boneh et al. [10] and Agrawal et al. [1].

Theorem 1. *If $\mathcal{SIG}$ is a strongly unforgeable signature scheme, $\mathcal{PKE}$ is a CCA2-secure public key encryption, then the above functional encryption construction $\mathcal{FE}$ is simulation-secure (Definition 6).*

Proof: We here prove that our scheme achieves the strong fully-adaptive simulation-security as defined in Definition 6. In order to prove the security, we need to construct a simulator Sim which interacts with an adversary $\mathcal{A}$. The ideal experiment $\mathbf{Expt}^{ideal}_{\mathcal{FE},\mathcal{A},\mathsf{Sim}}(1^\kappa)$ is as follows:

- Upon obtaining functionality F from the adversary, the simulator runs $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Gen}()$ and $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PKE.Gen}()$, and set $\mathsf{MPK} = (\mathsf{ek}, \mathsf{vk})$, and give MPK to the adversary. From now on, oracle access to the token will be simulated for the adversary.
- In the key generation, upon receiving the request on K from the adversary, the simulator computes $\sigma_K \leftarrow \mathsf{SIG.Sign}_{\mathsf{sk}}(K)$, and returns σ_K to the adversary. Note that now the simulator records (K, σ_K) into history.
- At any point when the adversary provides message M, the simulator is allowed to see the length $|M|$ and it is granted an oracle $F(\cdot, M)$. The simulator then computes $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(0^{|M|}; \omega)$ where ω is randomly

chosen, and sets ct as a pointer to the oracle $F(\cdot, M)$. The simulator records $(|M|, \mathsf{ct})$ into history, and returns ct to the adversary. If the same ct is recorded twice in the history, then the simulator returns `Abort`.

– At any point when the adversary queries the token with tuple $(\mathsf{ct}, K, \sigma_K)$, the simulator first checks if (K, σ_K) has been recorded in the history. If not, then it returns $\perp$. Else if the pair (K, σ_K) has been recorded, and ct has also been recorded in the history, then the simulator queries the corresponding oracle $F(\cdot, M)$, and learns $m = F(K, M)$. Then the simulator returns m to the adversary. Otherwise, if (K, σ_K) has been recorded but ct has not, the simulator computes $M \leftarrow \mathsf{PKE.Dec}_{\mathsf{dk}}(\mathsf{ct})$ and $m \leftarrow F(K, M)$, and returns m to the adversary.
– Let $M_1^*, \ldots, M_n^*$ be the messages that the adversary queried for ciphertexts. If the adversary finally outputs a value α, output $(\alpha, \{M_i^*\}_{i \in [n]})$.

From the above simulation, we can easily see that only valid users who participate in the key generation are able to use the token to decrypt ciphertexts. Furthermore, for a ciphertext $\mathsf{ct} = \mathsf{PKE.Enc}(M)$, the adversary cannot learn any extra information beyond $\{F(K_i, M)\}_i$ where $\{K_i\}_i$ have been registered in the key generation.

Next, we show that the ideal experiment is computationally close to the real experiment, by developing a sequence of hybrids between them.

Hybrid 0: This is the real experiment $\mathbf{Expt}^{real}_{\mathcal{FE},\mathcal{A}}(1^\kappa)$. As described in construction $\mathcal{FE}$, upon obtaining functionality F, we first generate $(\mathsf{MPK}, \mathsf{MSK}, \mathbf{T}) \leftarrow \mathsf{FE.Setup}(1^\kappa, F)$ where $\mathsf{MPK} = (\mathsf{ek}, \mathsf{vk})$ and $\mathsf{MSK} = \mathsf{sk}$, and give MPK to the adversary. At any moment when the adversary queries $\mathsf{FE.Key}()$ with K, we return $\mathsf{SK}_K = \sigma_K$ where $\sigma_K \leftarrow \mathsf{SIG.Sign}_{\mathsf{sk}}(K)$. At any point when the adversary outputs M_i^*, we return the adversary with $\mathsf{CT}_i^* = \mathsf{ct}_i^*$ where $\mathsf{ct}_i^* \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}}(M_i^*; \omega_i^*)$. At any point when the adversary queries the token with tuple $(\mathsf{ct}, K, \sigma_K)$, the token will behave as follows: if the pair (K, σ_K) is not verified, then return $\perp$. Otherwise if the pair is verified, i.e., $\mathsf{SIG.Vrfy}_{\mathsf{vk}}(K, \sigma_K) = 1$, then use dk to decrypt ct into $M \leftarrow \mathsf{PKE.Dec}_{\mathsf{dk}}(\mathsf{ct})$, and return $m = F(K, M)$. We then return m to the adversary. Let $M_1^*, \ldots, M_n^*$ be the values that the adversary queried for ciphertexts. If the adversary finally outputs a value α, we output $(\alpha, \{M_i^*\}_{i \in [n]})$.

Hybrid 1: This hybrid is the same as Hybrid 0 except the following: In this hybrid, we change the token's responses to the adversary. At any point when the adversary queries the token with tuple $(\mathsf{ct}, K, \sigma_K)$, if $\mathsf{SIG.Vrfy}_{\mathsf{vk}}(K, \sigma_K) = 1$ while the pair (K, σ_K) never appears in the queries to $\mathsf{FE.Key}()$, then the hybrid outputs `Abort`.

Hybrid 1 and Hybrid 0 are the same except that `Abort` occurs. Based on the strong unforgeability of $\mathcal{SIG}$, we claim the event of `Abort` occurs with negligible probability. Therefore, Hybrid 1 and Hybrid 0 are computationally indistinguishable. Towards contradiction, assume there is a distinguisher $\mathcal{A}$ can distinguish Hybrid 0 from Hybrid 1. We next show an algorithm $\mathcal{B}$ that breaks the strong unforgeability of $\mathcal{SIG}$ as follows:

– Upon receiving the encryption key vk, $\mathcal{B}$ internally simulates $\mathcal{A}$. $\mathcal{B}$ works the same as in Hybrid 0 except the following: At any point when the adversary provides functionality F, $\mathcal{B}$ computes $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PKE.Gen}()$, and sets $\mathsf{MPK} := (\mathsf{ek}, \mathsf{vk})$. At any moment when the adversary queries $\mathsf{FE.Key}()$ with K, $\mathcal{B}$ queries its own signing oracle with K and receives σ_K, and then $\mathcal{B}$ returns σ_K to $\mathcal{A}$ as the response.

  At any point when the adversary queries the token with tuple $(\mathsf{ct}, K, \sigma_K)$, if $\mathsf{SIG.Vrfy}_{\mathsf{vk}}(K, \sigma_K) = 1$, but (K, σ_K) never appears in the queries to $\mathsf{FE.Key}()$, then the event `Abort` occurs, $\mathcal{B}$ halts and output (K, σ_K) to its challenger as the forged pair.

We note that the view of the above simulated $\mathcal{A}$ is the same as that in Hybrid 1. We further note that as long as the event `Abort` does not occur, $\mathcal{A}$'s view is the same as that in Hybrid 0. Since $\mathcal{A}$ is able to distinguish the two hybrids, that means the event `Abort` will occur with non-negligible probability. That says, $\mathcal{B}$ is a successful unforgeability attacker against $\mathcal{SIG}$, which reaches a contradiction. Therefore, Hybrid 0 and Hybrid 1 are computationally indistinguishable.

Hybrid 2: This hybrid is the same as Hybrid 1 except the following: Whenever the adversary queries on M_i^*, we compute $\hat{\mathsf{ct}}_i^* \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}}(M_i^*; \omega_i^*)$, and record $(|M_i^*|, \hat{\mathsf{ct}}_i^*)$. Here we can easily simulate an oracle $F(\cdot, M_i^*)$ based on M_i^*, and we set the ciphertext ct^* as the pointer to the oracle. Furthermore, we change

the token's responses to the adversary. At any point when the adversary queries the token with tuple $(\mathsf{ct}, K, \sigma_K)$ where the pair (K, σ_K) has been recorded, we carry out the following: if ct has been recorded then we based on it query the corresponding oracle $F(\cdot, M_i^*)$ with K and receive $m = F(K, M_i^*)$. Then we return m to the adversary.

We can easily see that the views of $\mathcal{A}$ are the same in Hybrid 1 and Hybrid 2.

Hybrid 3.j, where $j = 0, \ldots, n$: Here n is the total number of messages the adversary has queried for ciphertexts. This hybrid is the same as Hybrid 2 except the following:

When the adversary queries on $\{M_i^*\}_{i \in [n]}$, the messages $\{M_1^*, \ldots, M_j^*\}$ are blocked; instead, we are allowed to see the length of the messages, i.e, $|M_1^*|, \ldots, |M_j^*|$, and have oracle access to $F(\cdot, M_1^*), \ldots, F(\cdot, M_j^*)$. Note that we are now still allowed to see the messages $\{M_{j+1}^*, \ldots, M_n^*\}$, and therefore we can easily simulate the oracles $F(\cdot, M_{j+1}^*), \ldots, F(\cdot, M_n^*)$.

We change the response to the adversary's query on $\{M_i^*\}_{i=1,\ldots,n}$. We now return the adversary with $\mathsf{CT}_i^* = \hat{\mathsf{ct}}_i^*$ for all $i \in [n]$. Here $\hat{\mathsf{ct}}_i^* \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}}(0^{|M_i^*|}; \omega_i^*)$ for all $i \in [1, \ldots, j]$, and $\hat{\mathsf{ct}}_i^* \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}}(M_i^*; \omega_i^*)$ for all $i \in [j+1, \ldots, n]$.

Based on the CCA2-security of $\mathcal{PKE}$, we claim Hybrid 3.j and Hybrid 3.$(j+1)$ are computationally indistinguishable for all $j = 0, \ldots, n-1$. Towards contradiction, assume there is a distinguisher $\mathcal{A}$ who can distinguish Hybrid 3.j from Hybrid 3.$(j+1)$. We next show an algorithm $\mathcal{B}$ that breaks the CCA2-security of $\mathcal{PKE}$ as follows:

- Upon receiving the encryption key ek, $\mathcal{B}$ internally simulates $\mathcal{A}$. $\mathcal{B}$ works the same as in Hybrid 3.j except the following:
  - Upon $\mathcal{A}$'s query on M_{j+1}^*, $\mathcal{B}$ queries LR-oracle LR with $(M_{j+1}^*, 0^{|M_{j+1}^*|})$; in turn it gets back a ciphertext ct_{j+1}^* which is $\mathsf{PKE.Enc}(0^{|M_{j+1}^*|})$ or $\mathsf{PKE.Enc}(M_{j+1}^*)$ from the LR-oracle.
  - Upon receiving $\mathcal{A}$'s query to the token with tuple $(\mathsf{ct}, K, \sigma_K)$ where (K, σ_K) has been recorded, if ct has been recorded then $\mathcal{B}$ simulates the corresponding oracle $F(\cdot, M_i^*)$ for K and provides $m = F(K, M_i^*)$ to $\mathcal{A}$. If ct has not been recorded, then $\mathcal{B}$ queries its decryption oracle to obtain the plaintext M of the ciphertext ct, and then return $m = F(K, M)$ to the adversary.
- Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Let β be the hidden bit associated with the LR oracle. We note that when $\beta = 0$, the algorithm $\mathcal{B}$ exactly simulates the Hybrid 3.j to $\mathcal{A}$; when $\beta = 1$, $\mathcal{B}$ simulates exactly the Hybrid 3.$(j+1)$ to $\mathcal{A}$. Under the assumption, since $\mathcal{A}$ is able to distinguish the two hybrids in non-negligible probability, that means the constructed $\mathcal{B}$ is successful CCA2 attacker against $\mathcal{PKE}$, which reaches a contradiction. Therefore Hybrid 3.j and Hybrid 3.$(j+1)$ are computationally indistinguishable.

Furthermore, we note that Hybrid 3.0 is the same as Hybrid 2, and Hybrid 3.n is the ideal experiment. Based on the above argument we already see the real experiment and the ideal experiment are in distinguishable. This means the construction $\mathcal{FE}$ is simulation secure as defined in Definition 6.

■

4.3 Token-based FE Construction — Solution #2

In our solution #1 presented in the previous section, the token size is linear of function F. Here we present our solution #2, a functional encryption scheme $\mathcal{FE} = \mathsf{FE}.\{\mathsf{Setup}, \mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}\}$ in the token model where the complexity of token is independent of the complexity of F. We use the following tools: FHE, digital signature, publicly verifiable SNARG, and simulation-extractable NIZK. (Please refer to Section 2 for the definitions.)

In the setup stage, the authority generates key pairs $(\mathsf{ek}, \mathsf{dk})$ and $(\mathsf{vk}, \mathsf{sk})$ for FHE and for digital signature respectively. The authority also sets up the reference strings crs and $(\mathsf{rs}, \mathsf{vrs})$ for NIZK and for SNARG respectively. Note that the reference string vrs for SNARG verification is very short and it is independent of F. The authority sets $(\mathsf{ek}, \mathsf{vk}, \mathsf{crs}, \mathsf{rs}, \mathsf{vrs})$ as its master public key MPK, and sk as the master secret key MSK. In addition, the authority initializes the token $\mathbf{T}$ with the public information $(\mathsf{ek}, \mathsf{vk}, \mathsf{crs}, \mathsf{vrs})$, and the secret decryption key dk.

The key generation stage is the same as that in the previous solution; for each user associated with a key K, the authority uses the MSK to generate a signature σ_K on the key K; in addition the authority sends the user an identical copy of the token. The encryption algorithm is different from that in the previous solution: To encrypt a message M, one takes two steps: (1) encrypt it using the FHE public key ek; that

- **Setup:** on input a security parameter 1^κ, a functionality $F \in \mathcal{F}_\kappa$, the setup algorithm $\mathsf{Setup}()$ performs the following steps to generate MPK, MSK, and a deterministic stateless token $\mathbf{T}$.
  - Execute $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{FHE.Gen}(1^\kappa)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Gen}(1^\kappa)$, and $\mathsf{crs} \leftarrow \mathsf{NIZK.Gen}(1^\kappa)$.
  - Define $\hat{F}(K, \mathsf{ct}) \triangleq \mathsf{FHE.Eval}_{\mathsf{ek}}(\mathsf{ct}, F(K, \cdot))$. Execute $(\mathsf{rs}, \mathsf{vrs}) \leftarrow \mathsf{SNARG.Gen}(1^\kappa, \hat{F})$.
  - Initiate a token $\mathbf{T}$ with values $(\mathsf{dk}, \mathsf{ek}, \mathsf{vk}, \mathsf{crs}, \mathsf{vrs})$.
  - Output $\mathsf{MPK} = (\mathsf{ek}, \mathsf{vk}, \mathsf{crs}, \mathsf{rs}, \mathsf{vrs})$, $\mathsf{MSK} = (\mathsf{sk})$.
- **Key Generation:** on input a master secret key MSK and a functionality key K, $\mathsf{Key}()$ generates SK_K as:
  - Execute $\sigma_K \leftarrow \mathsf{SIG.Sign}_{\mathsf{sk}}(K)$. Output $\mathsf{SK}_K = (\sigma_K)$.
- **Encryption:** on input a master public key MPK and a message M, the encryption algorithm $\mathsf{Enc}()$ generates CT as follows.
  - Execute $\mathsf{ct} \leftarrow \mathsf{FHE.Enc}_{\mathsf{ek}}(M; \omega)$, where ω is the randomness used in the encryption.
  - Execute $\pi \leftarrow \mathsf{NIZK}.P(\mathsf{crs}, (\mathsf{ek}, \mathsf{ct}), (M, \omega))$ with respect to the relation

$$R_{\mathsf{FHE}} = \{((\mathsf{ek}, \mathsf{ct}), (M, \omega)) : \mathsf{FHE.Enc}_{\mathsf{ek}}(M; \omega) = \mathsf{ct}\}.$$

  - Output $\mathsf{CT} = (\mathsf{ct}, \pi)$
- **Decryption:** on input SK_K and a ciphertext $\mathsf{CT} = (\mathsf{ct}, \pi)$ of a message M, with access to a token $\mathbf{T}$, the decryption algorithm $\mathsf{Dec}^{\mathbf{T}}()$ performs the following steps to decrypt $m = F(K, M)$:
  - Execute $(\tilde{\mathsf{ct}}, \varpi) \leftarrow \mathsf{SNARG}.P(\mathsf{rs}, (K, \mathsf{ct}))$. Here $\tilde{\mathsf{ct}} = \hat{F}(K, \mathsf{ct}) = \mathsf{FHE.Eval}_{\mathsf{ek}}(\mathsf{ct}, F(K, \cdot))$.
  - Query the token $m \leftarrow \mathbf{T}(\mathsf{CT}, K, \mathsf{SK}_K, \tilde{\mathsf{ct}}, \varpi)$. Output m.
- **Token Operations:** on query $(\mathsf{CT}, K, \mathsf{SK}_K, \tilde{\mathsf{ct}}, \varpi)$, where $\mathsf{CT} = (\mathsf{ct}, \pi)$ and $\mathsf{SK}_K = \sigma_K$, the token $\mathbf{T}$ carries out the following operations.
  - Execute $\mathsf{SIG.Vrfy}_{\mathsf{vk}}(K, \sigma_K)$, $\mathsf{NIZK}.V(\mathsf{crs}, (\mathsf{ek}, \mathsf{ct}), \pi)$, and $\mathsf{SNARG}.V(\mathsf{vrs}, (K, \mathsf{ct}), \tilde{\mathsf{ct}}, \varpi)$.
  - Return $\mathsf{FHE.Dec}_{\mathsf{dk}}(\tilde{\mathsf{ct}})$ if all above verifications accept, and return $\perp$ otherwise.

Fig. 2. Solution #2. Here $\mathsf{FHE}.\{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}\}$ is a fully homomorphic encryption, $\mathsf{SIG}.\{\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy}\}$ is a signature scheme, $\mathsf{SNARG}.\{\mathsf{Gen}, P, V\}$ is a SNARG scheme, $\mathsf{NIZK}.\{\mathsf{Gen}, P, V\}$ is a NIZK scheme.

is, the ciphertext is $\mathsf{ct} \leftarrow \mathsf{FHE.Enc}_{\mathsf{ek}}(M)$; (2) generate an NIZK that the ciphertext ct is honestly generated. The ciphertext for message M is (ct, π).

The decryption algorithm is different from that in the previous solution as well. Our goal as stated before is to obtain a solution in which the complexity of the token is independent of the complexity of F. The idea is to let the token to "outsource" most of the computation of F to the user. Concretely, to decrypt a ciphertext (ct, π), the user who is associated with key K computes the transformed ciphertext $\tilde{\mathsf{ct}}$ by homomorphically evaluating ct with $F(K, \cdot)$; to be sure that the transformation is carried out correctly, the user also provides a SNARG ϖ. Then the user queries the token $\mathbf{T}$ with an input tuple $(\mathsf{ct}, \pi, K, \sigma_K, \tilde{\mathsf{ct}}, \varpi)$; the token first verifies if signature, NIZK, SNARG are all valid; if so, the token decrypts the ciphertext $\tilde{\mathsf{ct}}$ into message m and returns m, and it returns $\perp$ otherwise. A formal description of our scheme can be found in Figure 2.

Note that, similar to solution #1, our scheme here also has succinct ciphertext size. Our ciphertext consists of an FHE ciphertext and an NIZK, both of which are independent of the complexity of F. On the other hand, here our token does *not* need to evaluate F to decrypt, and thus the complexity of the token is independent of the complexity of F.

Our scheme here also satisfies the strong fully-adaptive simulation-security as defined in Definition 4. The proof idea is very similar to that in the previous section, which crucially relies on the fact that in the simulation, the simulator gets to simulate token's answers to the queries made by the adversary. Next, we briefly highlight the differences between the two solutions. In both constructions, the user can only provide authenticated inputs to the hardware token, and digital signature is used to authenticate K. But two different approaches are used to authenticate the ciphertext: in solution #1, the authentication is guaranteed by the CCA2 security of the encryption, while in solution #2, the authentication is provided by the simulation-extractability of the NIZK, and the soundness of the SNARG.

Theorem 2. *If $\mathcal{SNARG}$ is a publicly verifiable SNARG scheme, $\mathcal{NIZK}$ is a zero-knowledge and simulation-extractable NIZK scheme, $\mathcal{SIG}$ is a strong unforgeable signature scheme, $\mathcal{FHE}$ is a secure fully homomorphic encryption scheme, then the above construction $\mathcal{FE}$ is simulation-secure functional encryption scheme.*

Proof can be found in the full version.

Acknowledgments. We would like to thank the anonymous reviewers for helpful feedback. We thank Vipul Goyal and Hoeteck Wee for helpful discussions.

References

1. S. Agrawal, S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption: New perspectives and lower bounds. Crypto 2013.
2. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.
3. M. Barbosa and P. Farshim. On the semantic security of functional encryption schemes. In *Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 143–161. Springer, 2013.
4. M. Bellare and A. O'Neill. Semantically secure functional encryption: Possibility results, impossibility results, and the quest for a general definition. Cryptology ePrint Archive, Report 2012/515, 2012. `http://eprint.iacr.org/`.
5. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. STOC 2013.
6. N. Bitansky, R. Canetti, S. Goldwasser, S. Halevi, Y. T. Kalai, and G. N. Rothblum. Program obfuscation with leaky hardware. In *Advances in Cryptology — Asiacrypt 2011*, vol. 7073 of *LNCS*, pages 722–739. Springer, 2011.
7. D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In R. Cramer, editor, *Advances in Cryptology — Eurocrypt 2005*, volume 3494 of *LNCS*, pages 440–456. Springer, 2005.
8. D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology — Crypto 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.
9. D. Boneh, A. Raghunathan, and G. Segev. Function-private identity-based encryption: Hiding the function in functional encryption. Crypto 2013.
10. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.
11. D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 535–554. Springer, 2007.
12. N. Chandran, V. Goyal, and A. Sahai. New constructions for UC secure computation using tamper-proof hardware. In *Advances in Cryptology — Eurocrypt 2008*, volume 4965 of *LNCS*, pages 545–562. Springer, 2008.
13. I. Damgård, J. B. Nielsen, and D. Wichs. Universally composable multiparty computation with partially isolated parties. In *Theory of Cryptography Conference*, volume 5444 of *LNCS*, pages 315–331. Springer, 2009.
14. A. De Caro, V. Iovino, A. Jain, A. O'Neill, O. Paneth, and G. Persiano. On the achievability of simulation-based security for functional encryption. Crypto 2013.
15. N. Döttling, D. Kraschewski, and J. Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In *Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 164–181. Springer, Mar. 2011.
16. N. Döttling, T. Mie, J. Müller-Quade, and T. Nilges. Implementing resettable UC functionalities with untrusted tamper-proof hardware-tokens. TCC 2013.
17. S. Garg, C. Gentry, S. Halevi, A. Sahai, and B. Waters. Attribute based encryption for circuits from multilinear maps. Crypto 2013.
18. S. Garg, C. Gentry, A. Sahai, and B. Waters. Witness encryption and its applications. STOC 2013.
19. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology — Crypto 2010*, vol. 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.
20. R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. Eurocrypt 2013.
21. C. Gentry. Fully homomorphic encryption using ideal lattices. In *41st Annual ACM Symposium on Theory of Computing*, pages 169–178. ACM Press, 2009.
22. C. Gentry and A. Silverberg. Hierarchical ID-based cryptography. In *Advances in Cryptology — Asiacrypt 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 548–566. Springer, 2002.
23. C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *43rd Annual ACM Symposium on Theory of Computing*, pages 99–108. ACM Press, 2011.
24. S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Succinct functional encryption and applications: Reusable garbled circuits and beyond. STOC 2013.

25. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption with bounded collusions via multi-party computation. Crypto 2012.
26. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute-based encryption for circuits. STOC 2013.
27. V. Goyal, Y. Ishai, M. Mahmoody, and A. Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In *Advances in Cryptology — Crypto 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 173–190. Springer, 2010.
28. V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. Founding cryptography on tamper-proof hardware tokens. In *Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2010.
29. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98. ACM Press, 2006.
30. J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *Advances in Cryptology — Asiacrypt 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 444–459. Springer, 2006.
31. J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Advances in Cryptology — Asiacrypt 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340. Springer, 2010.
32. J. Groth, R. Ostrovsky, and A. Sahai. Perfect non-interactive zero knowledge for NP. In *Advances in Cryptology — Eurocrypt 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358. Springer, 2006.
33. J. Horwitz and B. Lynn. Toward hierarchical identity-based encryption. In *Advances in Cryptology — Eurocrypt 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 466–481. Springer, 2002.
34. J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2007.
35. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *J. Cryptology*, 26(2): 191–224, 2013.
36. A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *Advances in Cryptology — Eurocrypt 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 62–91. Springer, 2010.
37. A. B. Lewko and B. Waters. Unbounded HIBE and attribute-based encryption. In *Advances in Cryptology — Eurocrypt 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 547–567. Springer, 2011.
38. H. Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. TCC 2012.
39. S. Micali. Computationally sound proofs. *SIAM J. Computing*, 30(4):1253–1298, 2000.
40. T. Moran and G. Segev. David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. Eurocrypt 2008.
41. A. O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. `http://eprint.iacr.org/`.
42. A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM Conference on Computer and Communications Security*, pages 463–472. ACM Press, 2010.
43. A. Sahai and B. R. Waters. Fuzzy identity-based encryption. In *Advances in Cryptology — Eurocrypt 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.
44. B. Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *Advances in Cryptology — Crypto 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 619–636. Springer, 2009.