Functional Signcryption: Notion, Construction, and Applications*

Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay

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

Abstract. Functional encryption (FE) enables sophisticated control over decryption rights in a multi-user scenario, while functional signature (FS) allows to enforce complex constraints on signing capabilities. This paper introduces the concept of *functional signcryption* (FSC) that aims to provide the functionalities of both FE and FS in an *unified cost-effective primitive*. FSC provides a solution to the problem of achieving confidentiality and authenticity simultaneously in digital communication and storage systems involving multiple users with better efficiency compared to a sequential implementation of FE and FS. We begin by providing formal definition of FSC and formulating its security requirements. Next, we present a generic construction of this challenging primitive that supports arbitrary polynomial-size signing and decryption functions from known cryptographic building blocks, namely, *indistinguishability obfuscation* (IO) and *statistically simulation-sound non-interactive zero-knowledge proof of knowledge* (SSS-NIZKPoK). Finally, we exhibit a number of representative applications of FSC: (I) We develop the *first* construction of *attribute-based signcryption* (ABSC) supporting signing and decryption policies representable by *general polynomial-size circuits* from FSC. (II) We show how FSC can serve as a tool for building SSS-NIZKPoK system and IO, a result which in conjunction with our generic FSC construction can also be interpreted as establishing an equivalence between FSC and the other two fundamental cryptographic primitives.

Keywords: functional signcryption, indistinguishability obfuscation, statistically simulation-sound non-interactive zero-knowledge proof of knowledge, polynomial-size circuits.

1 Introduction

Confidential as well as authenticated message transfer and storage has been one of the central focus of cryptography since years. In the public key setting, a standard approach for achieving this goal has been to utilize digital signature and public key encryption primitives in sequence. However, this strategy amounts to incurring a direct addition of the costs of both primitives. *Digital signcryption*, introduced by Zheng [Zhe97], is an ambitious cryptographic paradigm that unifies the functionalities of both encryption and authentication in a cost-effective formulation.

However, in the standard notion of digital signcryption, the control over signing and decryption rights is "all or nothing": Only those in possession of the secret signing key corresponding to the system public key can signcrypt a message and the resulting ciphertext can be unsigncrypted by only those having the matching secret decryption key. In the modern era of Internet communication and cloud technology where multiple users are involved, such an "all or nothing" control over signing and decryption capabilities is no longer sufficient, rather highly sophisticated restrictions over signing and decryption rights must be enforced.

In order to realize fine-grained control over decryption capabilities, the concept of *functional encryption* (FE) has been introduced [BSW11]. An FE scheme includes a trusted authority which holds a master secret key and publishes system public parameters. An encrypter uses this system public parameters to encrypt a message. A decrypter may obtain a decryption key $\text{DK}(g)$

* An extended abstract of this paper will appear in the proceedings of ProvSec 2015. This is the full version.

for some decryption function g from the authority if and only if the authority deems that the decrypter is entitled to possess that key. The decrypter can now use the decryption key $\mathrm{DK}(g)$ to decrypt a ciphertext encrypting some message m to obtain $g(m)$ and nothing more.

On the other hand, *functional signature* (FS), introduced in [BGI14], [BF14], allows managing complex signing credentials. Just like an FE scheme, an FS system also involves a trusted authority that publishes system public parameters and possesses a master signing key which can be used for signing any message and providing a constrained signing key $\mathrm{SK}(f)$ for some signing function f to a signer after verification of its signing credentials. This restricted signing key $\mathrm{SK}(f)$ can be used for producing signatures, verifiable under the system public parameters, on only those messages that are in the range of the function f.

In the past few years, a remarkable progress has taken place in the fields of FE and FS. In particular, FE and FS schemes supporting functions expressible in terms of *general polynomial-size circuits* have been invented based on advanced cryptographic primitives such as indistinguishability obfuscation, multilinear maps, statistically simulation-sound non-interactive zero-knowledge proof of knowledge, and so on [GGH$^+$13b], [ABG$^+$13], [Wat14], [GGHZ14], [BGI14], [BF14], [BMS13]. However, given this state of the art, exercising fine-grained control over the signing and decryption rights in a multi-user confidential and authenticated digital communication or storage system would necessitate implementing both FE and FS sequentially which would entail summing up the cost incurred by both primitives.

In this work, we put forward a *new* cryptographic paradigm termed as *functional signcryption* (FSC) that unifies the functionalities of both FE and FS. In other words, FSC aims to provide enhanced access control in the context of the traditional digital signcryption. FSC solves the issue of simultaneously managing signing and decryption credentials in a multi-user environment with better efficiency. More precisely, in an FSC scheme, we consider a trusted authority that holds a master secret key and publishes system public parameters. Using its master secret key, the authority can provide a signing key $\mathrm{SK}(f)$ for some signing function f to a signcrypter, as well as, a decryption key $\mathrm{DK}(g)$ corresponding to some decryption function g to a decrypter after verifying their credentials. Now such a signing key $\mathrm{SK}(f)$ enables a signcrypter to signcrypt, i.e., encrypt and authenticate simultaneously only those messages which are in the range of f, while a decryption key $\mathrm{DK}(g)$ can be utilized to unsigncrypt a ciphertext, which is the signcryption of some message m to retrieve $g(m)$ only and to verify the authenticity of the ciphertext at the same time.

We define two security notions for FSC: *message confidentiality* and *ciphertext unforgeability*. Roughly speaking, message confidentiality guarantees that arbitrary collusion of decrypters cannot retrieve any additional information about the signcrypted message from a ciphertext beyond the union of what they could obtain individually. On the other hand, ciphertext unforgeability assures that collusion of signcrypters cannot help them to generate a valid signcryption of a message which none of them could have signcrypted on their own.

A motivating practical application of FSC could be the following: Suppose the government of some country is collecting complete photographs of individuals as part of the census and storing the collected data in a large server to allow utilizing it in future by other organizations for various survey purposes. For maintaining the security and improving the quality of the collected photos at the same time, the government is using some photo-processing software that edits the photos and encrypts them before storing them to the server. Now, it is desirable that the software is allowed to perform only some minor touch-ups of the photos such as changing the color scale or removing red eyes, but is not allowed to make more significant changes such

as merging two photos or cropping a picture. FSC can naturally address this issue as follows: The government, acting as the trusted authority, would provide the photo-processing software (signcrypter) the signing keys ($\text{SK}(f)$) which allows it to signcrypt original photographs with only the allowable modifications (i.e., those in the range of f) and store the signcrypted photos in the server. Later, when some organization (decrypter) wants to access only those informations from stored photos meeting certain criteria (g), e.g., faces of individuals residing in a particular city, the government would give the organization the corresponding functional decryption key ($\text{DK}(g)$) after being fully convinced about the credentials of the organization. Now, when the organization would access that data base (i.e., signcryption of m) using the obtained decryption key, it could only obtain the face portion of the photographs of individuals living in that particular city ($g(m)$) and would be convinced that the photos obtained were undergone through only minor photo-editing modifications.

Our Contributions: We begin with formally introducing (FSC) and formalizing its security notions. We then present a *generic construction* of FSC that supports signing and decryption functions expressible as *general polynomial-size circuits*, assuming the existence of *indistinguishability obfuscation* (IO) for all polynomial-size circuits and *statistically simulation-sound non-interactive zero-knowledge proof of knowledge* (SSS-NIZKPoK) system for NP relations. Besides, we use ordinary public key encryption and digital signature schemes as building blocks for our FSC construction. We provide a rigorous security analysis of our FSC construction in our proposed security model and prove that it achieves *selective* message confidentiality against chosen plaintext attack (CPA), as well as, *selective* ciphertext unforgeability against chosen message attack (CMA). An IO $\mathcal{O}$ for a class of circuits $\mathbb{C}$ guarantees that given two equivalent circuits C_1 and C_2 from the class, the two distributions of obfuscations $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$ should be computationally indistinguishable. Very recently, few candidate constructions of IO have been proposed [GGH$^+$13b], [GLSW14], [PST14] based on multilinear maps [GGH13a], [CLT15]. On the other hand, an SSS-NIZKPoK system [Gro06], [GGH$^+$13b] for an NP relation R with associated language $\mathbb{L}$ is an extractable non-interactive proof system in which the proofs does not reveal any information about the witness to a computationally bounded adversary and it is infeasible to convince an honest verifier of a false statement even when the adversary is provided with a simulated proof.

Utilizing FSC, we further develop *attribute-based signcryption* (ABSC) supporting *arbitrary polynomial-size circuits*. ABSC is a related but weaker notion for controlling the signing and decryption capabilities in signcryption. ABSC has two variants, namely, *key-policy* and *ciphertext-policy*. Like FSC, in key-policy ABSC also there is a trusted authority who publishes system public parameters and uses a master secret key to produce signing and decryption keys corresponding to specific signing and decryption predicates. Now, the holder of such a signing key can signcrypt messages with respect to any set of decryption attributes and only those sets of signing attributes on which the predicate embedded in the signing key evaluates to 1. The signature and decryption attribute sets are attached in the clear with the ciphertext, so that anyone with a decryption key embedding a decryption predicate that outputs 1 on the associated decryption attribute set can verify the authenticity of the ciphertext with respect to the associated signing attribute set and retrieve the signcrypted message. In ciphertext-policy ABSC, the roles of predicates and attribute sets are reversed. Although in the last few years ABSC has gained a lot of attention in the literature [GNSN10], [RD14b], [RD14a], [WH11], the class of allowable signing and decryption predicates have been restricted to *monotone Boolean formulas* or, in other words, to *circuits with fan-out one*. As noted in [GGH$^+$13c], these schemes are vulnerable to "backtracking" attack. To the best of our knowledge, our proposed ABSC scheme is the *first* to realize general polynomial-size circuits for signing and decryption policies.

Finally, we establish an equivalence between FSC and the two primitives, namely, SSS-NIZKPoK for NP relations and IO for all polynomial-size circuits. Our generic construction, described in §4.1 shows that those cryptographic tools are *sufficient* to build FSC. A natural question that arises, therefore, is that whether SSS-NIZKPoK and IO are *necessary*, i.e., whether those are implied by FSC. In §6, we address this question by exhibiting that FSC indeed implies SSS-NIZKPoK for NP relations and IO for general polynomial-size circuits.

2 Preliminaries

Here we give the necessary background on the cryptographic primitives we will be using in our FSC construction. For positive integers n, a, b (with $a < b$), we let $[n] = \{1, \ldots, n\}$ and $[a, b] = \{a, \ldots, b\}$. For any set S, $x \leftarrow S$ represents the uniform random variable on S. For a randomized algorithm $\mathcal{M}$, we denote by $\theta = \mathcal{M}(\upsilon; r)$ the random variable defined by the output of $\mathcal{M}$ on input υ and randomness r, while $\theta \leftarrow \mathcal{M}(\upsilon)$ has the same meaning with the randomness suppressed. For any circuit C, $|C|$ denotes the size of C. For any two strings $s, s' \in \{0,1\}^*$, $s\|s'$ represents the concatenation of s and s'. A function ϵ is *negligible* if for every integer c, there exists an integer K such that for all $\lambda > K$, $|\epsilon(\lambda)| < 1/\lambda^c$.

2.1 Indistinguishability Obfuscation

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

Definition 1 (Indistinguishability Obfuscation: IO). *An indistinguishability obfuscator* (IO) $\mathcal{O}$ *for a circuit class* $\{\mathbb{C}_\lambda\}$ *is a probabilistic polynomial-time* (PPT) *uniform algorithm satisfying the following conditions:*

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

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

$$\mathsf{Pr}\big[(C_0, C_1, \alpha) \leftarrow \mathcal{D}_1(1^\lambda) \ : \ \forall \upsilon, C_0(\upsilon) = C_1(\upsilon)\big] > 1 - \epsilon(\lambda),$$
$$\textit{then } \big|\mathsf{Pr}[\mathcal{D}_2(\alpha, \mathcal{O}(1^\lambda, C_0)) = 1] - \mathsf{Pr}[\mathcal{D}_2(\alpha, \mathcal{O}(1^\lambda, C_1)) = 1]\big| < \epsilon(\lambda).$$

The circuit classes we are interested in are polynomial-size circuits, i.e., when $\mathbb{C}_\lambda$ is the collection of all circuits of size at most λ. This circuit class is denoted by P/poly. The first candidate construction of IO for P/poly was presented in [GGH+13b] in a generic model of encoded matrices. Later, [GLSW14], [PST14] have shown that IO for P/poly can be developed based on a single instance-independent assumption.

When clear from the context, we will drop 1^λ as an input to $\mathcal{O}$ and λ as a subscript of $\mathbb{C}$.

The GGHRSW Candidate IO Construction [GGH+13b]

In view of providing an overview of the design of this challenging cryptographic primitive, here we briefly discuss the IO construction of [GGH+13b]. However, this description is only a bird's eye-view with many technicalities omitted. In [GGH+13b], for building IO for P/poly, the authors proceed in two steps:

(I) **IO for $\mathbf{NC^1}$**: At first, they construct an IO for a restricted circuit class, namely, the family of circuits with logarithmic depth, polynomial size, and bounded fan-in. This circuit family is

denoted as NC^1. The construction utilizes a restricted version of multilinear maps [GGH13a], [CLT15], referred to as multilinear jigsaw puzzles in [GGH$^+$13b]. The celebrated theorem due to Barrington [Bar86] shows that any NC^1 circuit C with input length ζ and depth d can be transformed into an *oblivious matrix branching program*, i.e., it can be converted into a sequence $\text{BP} = \{(\mathsf{inp}(i), \boldsymbol{A}_{i,0}, \boldsymbol{A}_{i,1})\}_{i\in[k]}$ such that $k \leq 4^d$, where the function $\mathsf{inp} : [k] \to [\zeta]$ describes what input bit is examined at the i-th step and $\boldsymbol{A}_{i,\beta}$'s are 5×5 permutation matrices. On any particular ζ-bit input $\upsilon = \upsilon_1 \dots \upsilon_\zeta$, the evaluation of BP amounts to computing the product matrix $\boldsymbol{P} = \prod_{i\in[k]} \boldsymbol{A}_{i,\upsilon_{\mathsf{inp}(i)}}$, outputting 1 if $\boldsymbol{P}$ is the identity matrix $\boldsymbol{I}$ and 0 otherwise. With the above transformation in place, the obfuscation procedure now attempts to garble the branching program BP as described below. Let $N = 2k + 5$.

1. Run the instance generation routine of the multilinear jigsaw generator to get the underlying ring $\mathbb{Z}_p$, the public parameters PARAMS, and a secret state ψ to be passed to the encoding algorithm of the generator where we take the multilinearity level of the jigsaw puzzle to be $[k+2]$.

2. Sample random and independent scalers $\{\omega_{i,0}, \omega_{i,1}, \omega'_{i,0}, \omega'_{i,1}\}_{i\in[k]}$ from $\mathbb{Z}_p$, subject to the constraint that $\prod_{i\in I_t} \omega_{i,0} = \prod_{i\in I_t} \omega'_{i,0}$ and $\prod_{i\in I_t} \omega_{i,1} = \prod_{i\in I_t} \omega'_{i,1}$ for all $t \in [\zeta]$, where $I_t = \{i \ : \ i \in [k] \ \bigwedge \ \mathsf{imp}(i) = t\}$.

3. For every $i \in [k]$, compute two pairs of $(2N + 5) \times (2N + 5)$ block-diagonal matrices $\{\boldsymbol{D}_{i,\beta}\}_{i\in[k];\ \beta\in\{0,1\}}$, $\{\boldsymbol{D}'_{i,\beta}\}_{i\in[k];\ \beta\in\{0,1\}}$ where the diagonal entries $1, \dots, 2N$ are chosen at random and the bottom-right 5×5 are scaled $\boldsymbol{A}_{i,\beta}$'s and the scaled identity matrix respectively:

$$\boldsymbol{D}_{i,\beta} \sim \begin{pmatrix} \$ & & & \\ & \ddots & & \\ & & \$ & \\ & & & \omega_{i,\beta}\boldsymbol{A}_{i,\beta} \end{pmatrix}, \quad \boldsymbol{D}'_{i,\beta} \sim \begin{pmatrix} \$ & & & \\ & \ddots & & \\ & & \$ & \\ & & & \omega'_{i,\beta}\boldsymbol{I} \end{pmatrix},$$

   where the \$'s are random coefficients on main diagonal that are unique to each matrix and the unspecified entries are 0.

4. Choose two sets of vectors $\{\boldsymbol{s}, \boldsymbol{t}\}$ and $\{\boldsymbol{s}', \boldsymbol{t}'\}$ of dimension $2N + 5$ as follows: The entries $1, \dots, N$ in the $\boldsymbol{s}$ and $\boldsymbol{s}'$ vectors are set to 0, while the entries $N + 1, \dots, 2N$ are selected at random. In the $\boldsymbol{t}$ and $\boldsymbol{t}'$ vectors, the entries $1, \dots, N$ are chosen at random, while $N+1, \dots, 2N$ are set to 0. Next two pairs of random vectors $\{\boldsymbol{s}^*, \boldsymbol{t}^*\}$ and $\{\boldsymbol{s}'^*, \boldsymbol{t}'^*\}$ of length 5 are chosen such that $\langle \boldsymbol{s}^*, \boldsymbol{t}^* \rangle = \langle \boldsymbol{s}'^*, \boldsymbol{t}'^* \rangle$. The last 5 entries of $\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{s}', \boldsymbol{t}'$ are taken to be the entries of $\boldsymbol{s}^*, \boldsymbol{t}^*, \boldsymbol{s}'^*, \boldsymbol{t}'^*$ respectively.

5. Sample two sets of $(k+1)$ random full-rank $(2N+5)\times(2N+5)$ matrices over $\mathbb{Z}_p$, $\{\boldsymbol{R}_0, \dots, \boldsymbol{R}_k\}$ and $\{\boldsymbol{R}'_0, \dots, \boldsymbol{R}'_k\}$ and compute their inverses.

6. Compute the randomized branching program as follows:

$$\begin{aligned} &\text{RND}_p(\text{BP}) = \{\text{P-PROG}, \text{D-PROG}\} \\ &= \left\{ \begin{pmatrix} \widetilde{\boldsymbol{s}} = \boldsymbol{s}\boldsymbol{R}_0^{-1}, \widetilde{\boldsymbol{t}} = \boldsymbol{R}_k\boldsymbol{t}^\intercal, \\ \{\widetilde{\boldsymbol{D}}_{i,\beta} = \boldsymbol{R}_{i-1}\boldsymbol{D}_{i,\beta}\boldsymbol{R}_i^{-1}\}_{i\in[k];\beta\in\{0,1\}} \end{pmatrix}, \begin{pmatrix} \widetilde{\boldsymbol{s}}' = \boldsymbol{s}'(\boldsymbol{R}'_0)^{-1}, \widetilde{\boldsymbol{t}}' = \boldsymbol{R}'_k\boldsymbol{t}'^\intercal, \\ \{\widetilde{\boldsymbol{D}}'_{i,\beta} = \boldsymbol{R}'_{i-1}\boldsymbol{D}'_{i,\beta}(\boldsymbol{R}'_i)^{-1}\}_{i\in[k];\beta\in\{0,1\}} \end{pmatrix} \right\}, \end{aligned}$$

   where $\boldsymbol{u}^\intercal$ denotes transpose of the vector $\boldsymbol{u}$. Note that the above "randomized"program consists in essence of two parallel programs, namely, P-PROG and D-PROG. While P-PROG embeds

the original branching program BP with all the $\boldsymbol{A}_{i,\beta}$'s, D-PROG embeds a "dummy"program of the same length, consisting only of identity matrices (so it computes the constant function 1). During the evaluation of the obfuscated program, the dummy program is used for the purpose of equality test: The original program outputs 1 on a given input if and only if it agrees with the dummy program on that input.

7. Use the encoding algorithm of the multilinear jigsaw generator to encode each element of the step-i matrices $\widetilde{\boldsymbol{D}}_{i,\beta}, \widetilde{\boldsymbol{D}}'_{i,\beta}$ relative to the singleton index-set $i+1$, each elements of the vectors $\widetilde{\boldsymbol{s}}, \widetilde{\boldsymbol{s}}'$ relative to the singleton index-set $\{1\}$, and each element of the vectors $\widetilde{\boldsymbol{t}}, \widetilde{\boldsymbol{t}}'$ with respect to the singleton index-set $\{k+2\}$.

8. The obfuscated program consists of the public parameters PARAMS of the jigsaw generator plus all the encoded matrices and vectors.

The evaluation of the obfuscated program is carried out as follows: For any input $\upsilon = \upsilon_1 \dots \upsilon_\zeta$ to the original program, the corresponding matrices from both P-PROG and D-PROG are chosen and it is tested whether they yield the same result using only the allowed multilinear operations. More specifically, the evaluator computes encoding of $\widetilde{\boldsymbol{s}} \prod_{i\in[k]} \widetilde{\boldsymbol{D}}_{i,\upsilon_{\mathsf{inp}(i)}} \widetilde{\boldsymbol{t}} - \widetilde{\boldsymbol{s}}' \prod_{i\in[k]} \widetilde{\boldsymbol{D}}'_{i,\upsilon_{i,\mathsf{inp}(i)}} \widetilde{\boldsymbol{t}}'$ relative to the index set $[k+2]$ using multilinear operations and zero-test the result using the zero-testing algorithm of the zigsaw puzzle. If the zero-test passes, then the evaluator outputs 1, otherwise, it outputs 0.

The authors of [GGH+13b] prove that the above construction is indeed an IO for NC^1 circuits under a new complexity assumption whose validity they justify in a generic model of encoded matrices.

(II) **IO for $\mathsf{P/poly}$**: Having constructed an IO for NC^1, the authors proceed to build an IO for $\mathsf{P/poly}$. More precisely, they utilize the IO for NC^1 and (leveled) fully homomorphic encryption (FHE) [BGV12] with decryption in NC^1 to obtain IO for $\mathsf{P/poly}$. The key idea is to adopt the two-key paradigm [NY90] to work using IO's instead of witness-indistinguishable proofs.
To obfuscate a circuit $C \in \mathsf{P/poly}$ the following components are generated:

(a) Two public key-secret key pairs of the FHE scheme are selected and the public keys $\mathrm{PK}^{(1)}_{\mathsf{FHE}}$ and $\mathrm{PK}^{(2)}_{\mathsf{FHE}}$ (say) are published.
(b) Encryptions of the circuit C being obfuscated under both the FHE public keys $\mathrm{PK}^{(1)}_{\mathsf{FHE}}, \mathrm{PK}^{(2)}_{\mathsf{FHE}}$, yielding ciphertexts $\mathrm{CT}^{(1)}_{\mathsf{FHE}}, \mathrm{CT}^{(2)}_{\mathsf{FHE}}$ are generated.
(c) Finally, the IO of a certain NC^1 circuit $C_{\mathsf{DEC},0}$ constructed as follows is prepared.

The obfuscated circuit evaluator who holds an input υ, uses the FHE evaluation algorithm with input the FHE ciphertexts encrypting υ under $\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}$ and ciphertexts $\mathrm{CT}^{(1)}_{\mathsf{FHE}}, \mathrm{CT}^{(2)}_{\mathsf{FHE}}$ to yield encryptions of $C(\upsilon)$ under both $\mathrm{PK}^{(1)}_{\mathsf{FHE}}, \mathrm{PK}^{(2)}_{\mathsf{FHE}}$. Suppose these encryptions be denoted by $e^{(1)}_{\mathsf{FHE}}$ and $e^{(2)}_{\mathsf{FHE}}$ respectively. The evaluator also keeps track of all the intermediate bit values encountered during evaluation of the $e^{(1)}_{\mathsf{FHE}}$ and $e^{(2)}_{\mathsf{FHE}}$ which can be seen as a "proof" π that it used υ to perform the evaluation correctly on both $\mathrm{CT}^{(1)}_{\mathsf{FHE}}$ and $\mathrm{CT}^{(2)}_{\mathsf{FHE}}$. The evaluator then feeds $(e^{(1)}_{\mathsf{FHE}}, e^{(2)}_{\mathsf{FHE}}, \upsilon, \pi)$ into the obfuscated form of the circuit $C_{\mathsf{DEC},0}$. This circuit $C_{\mathsf{DEC},0}$ first checks the proof π to make sure that $e^{(1)}_{\mathsf{FHE}}$ and $e^{(2)}_{\mathsf{FHE}}$ were correctly computed. Note that this can be done easily in NC^1. In fact, since the evaluator has included all the intermediate bit values encountered during evaluation, so the circuit merely needs to check that each appropriate triple of intermediate bit values respects the corresponding gate operations used during the evaluation process. If the proof checks out, then the circuit $C_{\mathsf{DEC},0}$ decrypts $e^{(1)}_{\mathsf{FHE}}$ using the secret FHE key $\mathrm{SK}^{(1)}_{\mathsf{FHE}}$ corresponding to $\mathrm{PK}^{(1)}_{\mathsf{FHE}}$ and outputs this decrypted value which should be $C(\upsilon)$.

The principal insight behind the proof that the above construction indeed serves as an IO is that there is another NC^1 circuit $C_{\mathsf{DEC},1}$ that is equivalent to $C_{\mathsf{DEC},0}$, which simply decrypts $e^{(2)}_{\mathsf{FHE}}$ using the secret key $\mathrm{SK}^{(2)}_{\mathsf{FHE}}$ corresponding to $\mathrm{PK}^{(2)}_{\mathsf{FHE}}$ instead provided the proof π is verified. Because of the proof π that must be provided, both of these circuits always behave identically on all inputs. Furthermore, when using $C_{\mathsf{DEC},0}$, we note that $\mathrm{SK}^{(2)}_{\mathsf{FHE}}$ is never used anywhere, and, therefore, the simantic security of the FHE scheme using $\mathrm{PK}^{(2)}_{\mathsf{FHE}}$ is maintained even given $C_{\mathsf{DEC},0}$. Thus, by alternatively applying the simantic security of the FHE scheme and switching back and forth between $C_{\mathsf{DEC},0}$ and $C_{\mathsf{DEC},1}$ using the IO property, it is proven in [GGH+13b] that the obfuscator presented above is indeed an IO for $\mathsf{P/poly}$.

2.2 Statistically Simulation-Sound Non-Interactive Zero-Knowledge Proof of Knowledge

Simulation-sound non-interactive zero-knowledge proof of knowledge have been introduced and formalized in the full version of [Gro06]. However, here we slightly simplify the original definition following [GGH+13b] and [BGI14].

Definition 2 (Statistically Simulation-Sound Non-Interactive Zero-Knowledge Proof of Knowledge: SSS-NIZKPoK). *Let $R \subset \{0,1\}^* \times \{0,1\}^*$ be an NP (binary) relation. For pairs $(X, W) \in R$, we call X the statement and W the witness. Let $\mathbb{L} \subset \{0,1\}^*$ be the language consisting of statements in R. An $\mathsf{SSS\text{-}NIZKPoK}$ system for $\mathbb{L}$ consists of the following PPT algorithms:*

$\mathsf{SSS\text{-}NIZKPoK.Setup}(1^\lambda)$: *The trusted authority takes as input a security parameter 1^λ and publishes a common reference string* CRS.

$\mathsf{SSS\text{-}NIZKPoK.Prove}(\mathrm{CRS}, X, W)$: *Taking as input the common reference string* CRS*, a statement $X \in \mathbb{L}$ along with a witness W, a prover outputs a proof π for X.*

$\mathsf{SSS\text{-}NIZKPoK.Verify}(\mathrm{CRS}, X, \pi)$: *On input the common reference string* CRS*, a statement $X \in \{0,1\}^*$, and a proof π, a verifier outputs 1, if the proof π is acceptable, or 0, otherwise.*

$\mathsf{SSS\text{-}NIZKPoK.SimSetup}(1^\lambda, X)$: *The simulator takes as input the security parameter 1^λ together with a statement $X \in \{0,1\}^*$. It produces a simulated common reference string* CRS *along with a trapdoor* TR *that enables it to simulate a proof for X without access to a witness.*

$\mathsf{SSS\text{-}NIZKPoK.SimProve}(\mathrm{CRS}, \mathrm{TR}, X)$: *Taking as input the simulated common reference string* CRS*, the trapdoor* TR*, and the statement $X \in \{0,1\}^*$ for which* CRS *and* TR *have been generated, the simulator outputs a simulated proof π for X.*

$\mathsf{SSS\text{-}NIZKPoK.ExtSetup}(1^\lambda)$: *The extractor, on input the security parameter 1^λ, outputs an extraction-enabling common reference string* CRS *and an extraction trapdoor* $\widehat{\mathrm{TR}}$.

$\mathsf{SSS\text{-}NIZKPoK.Extr}(\mathrm{CRS}, \widehat{\mathrm{TR}}, X, \pi)$: *The extractor takes as input the extraction-enabling common reference string* CRS*, the extraction trapdoor $\widehat{\mathrm{TR}}$, a statement $X \in \{0,1\}^*$, and a proof π. It outputs a witness W.*

An $\mathsf{SSS\text{-}NIZKPoK}$ *system should possess the following properties*:

- ***Perfect Completeness***: *An* $\mathsf{SSS\text{-}NIZKPoK}$ *system is said to be perfectly complete if for all security parameter λ, all $X, W \in R$, all* $\mathrm{CRS} \leftarrow \mathsf{SSS\text{-}NIZKPoK.Setup}(1^\lambda)$*, and all* $\pi \leftarrow \mathsf{SSS\text{-}NIZKPoK.Prove}(\mathrm{CRS}, X, W)$*, we have* $\mathsf{SSS\text{-}NIZKPoK.Verify}(\mathrm{CRS}, X, \pi) = 1$.

- ***Statistical Soundness***: *An* SSS-NIZKPoK *system is statistically sound if for all non-uniform adversaries $\mathcal{A}$ there exists a negligible function ϵ such that for any security parameter λ, we have*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SSS\text{-}NIZKPoK,SOUND}}(\lambda) = \mathsf{Pr}\big[\mathrm{CRS} \leftarrow \mathsf{SSS\text{-}NIZKPoK.Setup}(1^\lambda);\ (X,\pi) \leftarrow \mathcal{A}(\mathrm{CRS})\ : \\ \mathsf{SSS\text{-}NIZKPoK.Verify}(\mathrm{CRS}, X, \pi) = 1 \bigwedge X \notin \mathbb{L}\big] < \epsilon(\lambda).$$

- ***Computational Zero-Knowledge***: *We define the* SSS-NIZKPoK *system to be computationally zero-knowledge if for all non-uniform* PPT *adversaries $\mathcal{A}$ there exists a negligible function ϵ such that for any security parameter λ, we have for all $X \in \mathbb{L}$*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SSS\text{-}NIZKPoK,ZK}}(\lambda) = \\ \big|\mathsf{Pr}\big[\mathrm{CRS} \leftarrow \mathsf{SSS\text{-}NIZKPoK.Setup}(1^\lambda); \pi \leftarrow \mathsf{SSS\text{-}NIZKPoK.Prove}(\mathrm{CRS}, X, W) : \mathcal{A}(\mathrm{CRS}, X, \pi) = 1\big] \\ - \mathsf{Pr}\big[(\mathrm{CRS}, \mathrm{TR}) \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimSetup}(1^\lambda, X); \pi \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimProve}(\mathrm{CRS}, \mathrm{TR}, X) : \\ \mathcal{A}(\mathrm{CRS}, X, \pi) = 1\big]\big| < \epsilon(\lambda)$$

where W is a witness corresponding to X.

- ***Knowledge Extraction***: *We call an* SSS-NIZKPoK *system a proof of knowledge for R if for any security parameter λ the following holds: For all non-uniform adversaries $\mathcal{A}$ there exists a negligible function ϵ_1 such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SSS\text{-}NIZKPoK,CRS}}(\lambda) = \big|\mathsf{Pr}\big[\mathrm{CRS} \leftarrow \mathsf{SSS\text{-}NIZKPoK.Setup}(1^\lambda) : \mathcal{A}(\mathrm{CRS}) = 1\big] \\ - \mathsf{Pr}\big[(\mathrm{CRS}, \widehat{\mathrm{TR}}) \leftarrow \mathsf{SSS\text{-}NIZKPoK.ExtSetup}(1^\lambda) : \mathcal{A}(\mathrm{CRS}) = 1\big]\big| < \epsilon_1(\lambda)$$

and for all non-uniform PPT *adversaries $\mathcal{A}$ there exists a negligible function ϵ_2 such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SSS\text{-}NIZKPoK,EXT}}(\lambda) = \mathsf{Pr}\big[(X,\pi) \leftarrow \mathcal{A}(\mathrm{CRS});\ W^* \leftarrow \mathsf{SSS\text{-}NIZKPoK.Extr}(\mathrm{CRS}, \widehat{\mathrm{TR}}, X, \pi)\ : \\ \mathsf{SSS\text{-}NIZKPoK.Verify}(\mathrm{CRS}, X, \pi) = 1 \bigwedge (X, W^*) \notin R\big] < \epsilon_2(\lambda).$$

- ***Statistical Simulation-Soundness***: *An* SSS-NIZKPoK *system is statistically simulation-sound if for all non-uniform adversaries $\mathcal{A}$ there exists a negligible function ϵ such that for any security parameter λ, we have for all statements $X \in \{0,1\}^*$*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SSS\text{-}NIZKPoK,SIM\text{-}SOUND}}(\lambda) = \\ \mathsf{Pr}\big[(\mathrm{CRS}, \mathrm{TR}) \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimSetup}(1^\lambda, X); \\ \pi \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimProve}(\mathrm{CRS}, \mathrm{TR}, X);\ (X^*, \pi^*) \leftarrow \mathcal{A}(\mathrm{CRS}, X, \pi)\ : \\ X^* \neq X \bigwedge X^* \notin \mathbb{L} \bigwedge \mathsf{SSS\text{-}NIZKPoK.Verify}(\mathrm{CRS}, X^*, \pi^*) = 1\big] < \epsilon(\lambda).$$

Realizing SSS-NIZKPoK

Constructions of *non-interactive zero-knowledge proof of knowledge* (NIZKPoK) for NP relations are well-known [GOS06], [GOS12]. Using any NIZKPoK together with a *non-interactive perfectly binding commitment* scheme we can construct an SSS-NIZKPoK system following the same technique as described in [GGH+13b]. For the sake of completeness we briefly sketch the construction here.

Recall that an NIZKPoK system consists of similar algorithms as defined above for SSS-NIZKPoK with the only exception that it lacks the statistical simulation-soundness feature possessed by SSS-NIZKPoK. On the other hand, a non-interactive perfectly binding commitment scheme involves a commitment function $\mathsf{Com}(\cdot;\cdot)$ which takes as input a message X along with randomness r and outputs a commitment $c = \mathsf{Com}(X;\ r)$ of X. The commitment scheme must

satisfy two properties, namely, *computational hiding* and *perfect binding*. Computational hiding guarantees that no computationally bounded adversary can distinguish as to which message X is locked in a commitment c, while perfect binding ensures that even an unbounded adversary cannot open a commitment c in two different ways.

Let R be an NP relation with associated language $\mathbb{L}$, κ be an upper bound on the length of the statements in $\mathbb{L}$, and 0^κ represents a special statement outside $\mathbb{L}$. Consider a non-interactive perfectly binding commitment scheme with commitment function $\mathsf{Com}(\cdot;\cdot)$ for message space $\{0,1\}^\kappa$ along with randomness space $\{0,1\}^\eta$, and an $\mathsf{NIZKPoK}$ system $\mathsf{NIZKPoK}$=($\mathsf{NIZKPoK.Setup}$, $\mathsf{NIZKPoK.Prove}$, $\mathsf{NIZKPoK.Verify}$, $\mathsf{NIZKPoK.SimSetup}$, $\mathsf{NIZKPoK.SimProve}$, $\mathsf{NIZKPoK.ExtSetup}$, $\mathsf{NIZKPoK.Extr}$) for the relation R' whose statements are of the form $X' = (X, c)$, witnesses are of the form $W' = (W, r)$, and

$$(X', W') \in R' \iff (X, W) \in R \bigvee c = \mathsf{Com}(X; r). \tag{1}$$

The $\mathsf{SSS\text{-}NIZKPoK}$ system is described as follows:

$\mathsf{SSS\text{-}NIZKPoK.Setup}(1^\lambda)$: The trusted authority generates $\text{CRS}' \leftarrow \mathsf{NIZKPoK.Setup}(1^\lambda)$ and $c = \mathsf{Com}(0^\kappa;\ r)$ using randomness $r \leftarrow \{0,1\}^\eta$. It publishes the common reference string $\text{CRS} = (\text{CRS}', c)$.

$\mathsf{SSS\text{-}NIZKPoK.Prove}(\text{CRS} = (\text{CRS}', c), X, W)$: A prover computes $\pi' \leftarrow \mathsf{NIZKPoK.Prove}(\text{CRS}', X' = (X, c), W' = (W, r))$ where r is some arbitrary random value in $\{0,1\}^\eta$. It outputs the proof $\pi = \pi'$.

$\mathsf{SSS\text{-}NIZKPoK.Verify}(\text{CRS} = (\text{CRS}', c), X, \pi = \pi')$: The verifier runs $\mathsf{NIZKPoK.Verify}(\text{CRS}', X' = (X, c), \pi')$ and outputs the result.

$\mathsf{SSS\text{-}NIZKPoK.SimSetup}(1^\lambda, X)$: The simulator generates $\text{CRS}' \leftarrow \mathsf{NIZKPoK.Setup}(1^\lambda)$ and $c = \mathsf{Com}(X; r)$ using randomness $r \leftarrow \{0,1\}^\eta$. It outputs the simulated common reference string as $\text{CRS} = (\text{CRS}', c)$. Its simulation trapdoor is $\text{TR} = r$.

$\mathsf{SSS\text{-}NIZKPoK.SimProve}(\text{CRS} = (\text{CRS}', c), \text{TR} = r, X)$: The simulator generates the proof $\pi' \leftarrow \mathsf{NIZKPoK.Prove}(\text{CRS}', X' = (X, c), W' = (W, r))$, where W is any arbitrary string, and outputs the simulated proof as $\pi = \pi'$.

$\mathsf{SSS\text{-}NIZKPoK.ExtSetup}(1^\lambda)$: The extractor generates $(\text{CRS}', \widehat{\text{TR}}') \leftarrow \mathsf{NIZKPoK.ExtSetup}(1^\lambda)$ and $c = \mathsf{Com}(0^\kappa; r)$ using randomness $r \leftarrow \{0,1\}^\eta$. It outputs the extraction enabling common reference string as $\text{CRS} = (\text{CRS}', c)$. Its extraction trapdoor is $\widehat{\text{TR}} = \widehat{\text{TR}}'$.

$\mathsf{SSS\text{-}NIZKPoK.Extr}(\text{CRS} = (\text{CRS}', c), \widehat{\text{TR}} = \widehat{\text{TR}}', X, \pi = \pi')$: The extractor executes the algorithm $\mathsf{NIZKPoK.Extr}(\text{CRS}', \widehat{\text{TR}}', X' = (X, c), \pi')$. If it obtains (W, r) for some strings W and r, then it outputs W. Otherwise, it outputs $\perp$ indicating failure.

The fact that the above construction satisfies all the properties of an $\mathsf{SSS\text{-}NIZKPoK}$ system can be observed as follows:

- **Perfect Completeness**: This property follows directly from that of the $\mathsf{NIZKPoK}$ system.
- **Statistical Soundness**: This follows straightway from that of the $\mathsf{NIZKPoK}$ system and the perfect binding property of the commitment scheme.
- **Computational Zero-Knowledge**: The computational zero-knowledge property of the above $\mathsf{SSS\text{-}NIZKPoK}$ system follows from the following hybrid argument. Consider the sequence of hybrid experiments described below:

– Hyb_0: This hybrid corresponds to the honest generation of the common reference string and proof.

– Hyb_1: In this hybrid, we construct the common reference string as is done in SSS-NIZKPoK.SimSetup. However, the proof is computed honestly with respect to the simulated common reference string.

– Hyb_2: Here, we generate the common reference string as in SSS-NIZKPoK.SimSetup and compute the proof as in SSS-NIZKPoK.SimProve. Note that this hybrid coincides with the simulation scenario.

Observe that, computational indistinguishability between Hyb_0 and Hyb_1 follows from the computational hiding property of the commitment scheme, while that between hybrids Hyb_1 and Hyb_2 follows from the computational zero-knowledge property of the NIZKPoK system.

- **Knowledge Extraction**: The common reference strings generated by SSS-NIZKPoK.Setup and that outputted by SSS-NIZKPoK.ExtSetup are statistically close as those produced by NIZKPoK.Setup and NIZKPoK.ExtSetup are statistically close. The extraction property follows from the extraction property of NIZKPoK system and the perfect binding property of the commitment scheme.

- **Statistical Simulation Soundness**: This follows directly from the statistical soundness of the NIZKPoK system and the fact that, in the simulation scenario, the only false statement for which an acceptable proof exists is the statement on which the simulated proof is provided since the commitment scheme is perfectly binding.

3 The Notion of Functional Signcryption

We now give a formal definition of a functional signcryption (FSC) scheme and explain in more detail the security requirements an FSC scheme must satisfy.

- **Syntax**: A functional signcryption (FSC) scheme for a message space $\mathbb{M}$, a family of signing functions $\mathbb{F} = \{f : \mathbb{D}_f \to \mathbb{M}\}$, and a class of decryption functions $\mathbb{G} = \{g : \mathbb{M} \to \mathbb{R}_g\}$, where $\mathbb{D}_f$ and $\mathbb{R}_g$ denote the domain of the function f and range of the function g respectively, consists of the following PPT algorithms:

  FSC.Setup(1^λ): The trusted authority takes as input the security parameter 1^λ and publishes the public parameters MPK, while keeps the master secret key MSK to itself.

  FSC.SKeyGen$(\text{MPK}, \text{MSK}, f)$: Taking as input the public parameters MPK, the master secret key MSK, and a signing function $f \in \mathbb{F}$ from a signcrypter, the trusted authority provides a signing key $\text{SK}(f)$ to the signcrypter.

  FSC.Signcrypt$(\text{MPK}, \text{SK}(f), z)$: A signcrypter takes as input the public parameters MPK, its signing key $\text{SK}(f)$ corresponding to some signing function $f \in \mathbb{F}$, and an input $z \in \mathbb{D}_f$. It produces a ciphertext CT which is a signcryption of $f(z) \in \mathbb{M}$.

  FSC.DKeyGen$(\text{MPK}, \text{MSK}, g)$: On input the public parameters MPK, the master secret key MSK, and a decryption function $g \in \mathbb{G}$ from a decrypter, the trusted authority hands the decryption key $\text{DK}(g)$ to the decrypter.

  FSC.Unsigncrypt$(\text{MPK}, \text{DK}(g), \text{CT})$: A decrypter, on input the public parameters MPK, its decryption key $\text{DK}(g)$ associated with its decryption function $g \in \mathbb{G}$, and a ciphertext CT

signcrypting a message $m \in \mathbb{M}$, attempts to unsigncrypt the ciphertext CT and outputs $g(m)$, if successful, or a special string $\perp$ indicating failure, otherwise.

- **Correctness**: An FSC scheme is correct if for all $f \in \mathbb{F}$, $z \in \mathbb{D}_f$, and $g \in \mathbb{G}$,

$$\mathsf{Pr}\big[(\text{MPK}, \text{MSK}) \leftarrow \mathsf{FSC.Setup}(1^\lambda) \ : \ \mathsf{FSC.Unsigncrypt}\big(\text{MPK}, \mathsf{FSC.DKeyGen}(\text{MPK}, \text{MSK}, g), \mathsf{FSC.Signcrypt}\big(\text{MPK}, \mathsf{FSC.SKeyGen}(\text{MPK}, \text{MSK}, f), z\big)\big) = g(f(z))\big] > 1 - \epsilon(\lambda)$$

for some negligible function ϵ.

- **Security**: An FSC scheme has two security requirements, namely, (I) *message confidentiality* and (II)*ciphertext unforgeability* which are described below. For simplicity, we present our security definitions for the *selective* model, where the adversary must decide the challenge messages up front, before the system parameters are chosen.

(I) ***message confidentiality***: We define this security notion on indistinguishability of ciphertexts against *chosen plaintext attack* (CPA) through the following game between a probabilistic adversary $\mathcal{A}$ and a probabilistic challenger $\mathcal{C}$.

Init: $\mathcal{A}$ submits two pairs $(f_0^*, z_0^*), (f_1^*, z_1^*)$ of signing functions and inputs in the respective domains that will be used to frame the challenge.

Setup: $\mathcal{C}$ performs $\mathsf{FSC.Setup}(1^\lambda)$ to obtain (MPK, MSK) and hands MPK to $\mathcal{A}$.

Query Phase 1: $\mathcal{A}$ may adaptively make any polynomial number of queries which may be of the following types to be answered by $\mathcal{C}$.

- *Signing key query*: Upon receiving a signing key query corresponding to a signing function $f \in \mathbb{F}$ from $\mathcal{A}$, $\mathcal{C}$ returns $\text{SK}(f)$ to $\mathcal{A}$ by running $\mathsf{FSC.SKeyGen}(\text{MPK}, \text{MSK}, f)$.
- *Decryption key query*: When $\mathcal{A}$ queries a decryption key for a decryption function $g \in \mathbb{G}$ to $\mathcal{C}$ subject to the constraint that $g(f_0^*(z_0^*)) = g(f_1^*(z_1^*))$, $\mathcal{C}$ provides the decryption key $\text{DK}(g)$ to $\mathcal{A}$ by executing $\mathsf{FSC.DKeyGen}(\text{MPK}, \text{MSK}, g)$.
- *Signcryption query*: In response to a signcryption query made by $\mathcal{A}$ for a signing function $f \in \mathbb{F}$ and an input $z \in \mathbb{D}_f$, $\mathcal{C}$ sends the ciphertext CT to $\mathcal{A}$, which is a signcryption of $f(z)$, by executing $\mathsf{FSC.Signcrypt}\big(\text{MPK}, \mathsf{FSC.SKeyGen}(\text{MPK}, \text{MSK}, f), z\big)$.

Challenge: $\mathcal{C}$ flips a random coin $b \leftarrow \{0, 1\}$ and generates the challenge ciphertext CT^* by executing $\mathsf{FSC.Signcrypt}\big(\text{MPK}, \mathsf{FSC.SKeyGen}(\text{MPK}, \text{MSK}, f_b^*), z_b^*\big)$.

Query Phase 2: $\mathcal{A}$ may continue adaptively to make a polynomial number of queries as in **Query Phase 1**, subject to the same restriction as earlier, and $\mathcal{C}$ provides the answer to them.

Guess: $\mathcal{A}$ eventually outputs a guess b' for b and wins the game if $b' = b$.

Definition 3. *An* FSC *scheme is defined to be selectively message confidential against* CPA *if for all* PPT *adversaries* $\mathcal{A}$ *there exists a negligible function* ϵ *such that for any security parameter* λ*,*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FSC,s\text{-}IND\text{-}CPA}}(\lambda) = |\mathsf{Pr}[b' = b] - 1/2| < \epsilon(\lambda).$$

(II) ***Ciphertext Unforgeability***: This notion of security is defined on existential unforgeability against *chosen message attack* (CMA) through the following game between a probabilistic adversary $\mathcal{A}$ and a probabilistic challenger $\mathcal{C}$.

Init: $\mathcal{A}$ declares a message $m^* \in \mathbb{M}$ to $\mathcal{C}$ on which the forgery will be outputted.

Setup: $\mathcal{C}$ runs $\mathsf{FSC.Setup}(1^\lambda)$ to obtain (MPK, MSK) and hands MPK to $\mathcal{A}$.

Query Phase: $\mathcal{A}$ may adaptively make a polynomial number of queries of the following types to $\mathcal{C}$ and $\mathcal{C}$ provides the answer to those queries.

- *Signing key query*: Upon receiving a signing key query from $\mathcal{A}$ corresponding to a signing function $f \in \mathbb{F}$ subject to the constraint that there exists no $z \in \mathbb{D}_f$ such that $f(z) = m^*$, $\mathcal{C}$ returns $\text{SK}(f)$ to $\mathcal{A}$ by executing $\mathsf{FSC.SKeyGen}(\text{MPK}, \text{MSK}, f)$.

- *Decryption key query*: When $\mathcal{A}$ queries a decryption key for a decryption function $g \in \mathbb{G}$, $\mathcal{C}$ gives $\text{DK}(g)$ to $\mathcal{A}$ by performing $\mathsf{FSC.DKeyGen}(\text{MPK}, \text{MSK}, g)$.

- *Signcryption query*: In response to a signcryption query of $\mathcal{A}$ corresponding to a signing function $f \in \mathbb{F}$ and input $z \in \mathbb{D}_f$, $\mathcal{C}$ returns the ciphertext CT, which is a signcryption of $f(z)$, to $\mathcal{A}$ by performing $\mathsf{FSC.Signcrypt}(\text{MPK}, \mathsf{FSC.SKeyGen}(\text{MPK}, \text{MSK}, f), z)$.

- *Unsigncryption query*: Upon receiving an unsigncryption query form $\mathcal{A}$ for a ciphertext CT under a decryption function $g \in \mathbb{G}$, $\mathcal{C}$ performs $\text{DK}(g) \leftarrow \mathsf{FSC.DKeyGen}(\text{MPK}, \text{MSK}, g)$ followed by $\mathsf{FSC.Unsigncrypt}(\text{MPK}, \text{DK}(g), \text{CT})$, and sends the result to $\mathcal{A}$.

Forgery: $\mathcal{A}$ eventually outputs a forgery CT^* on m^*. $\mathcal{A}$ wins the game if CT^* is indeed a valid functional signcryption of m^*, i.e., $\mathsf{FSC.Unsigncrypt}(\text{MPK}, \text{DK}(g), \text{CT}^*) = g(m^*)$ for all $g \in \mathbb{G}$, and there does not exist any (f, z) pair such that (f, z) was a signcryption query of $\mathcal{A}$ and $m^* = f(z)$.

Definition 4. *An* FSC *scheme is defined to be selectively ciphertext unforgeable against* CMA *if for all* PPT *adversaries* $\mathcal{A}$ *there exists a negligible function* ϵ *such that for any security parameter* λ*,*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FSC,s\text{-}UF\text{-}CMA}}(\lambda) = \mathsf{Pr}[\mathcal{A} \textit{ wins}] < \epsilon(\lambda).$$

4 Our FSC Scheme

In this section, we present our generic construction and proof of security of an FSC scheme supporting signing and decryption functions in P/poly from IO for P/poly and SSS-NIZKPoK for NP. In addition to these primitives, our construction utilizes public key encryption secure against chosen plaintext attack (CPA) and signature scheme with existential unforgeability against chosen message attack (CMA). Background material on these latter primitives can be found in [KL07].

Let λ be the underlying security parameter. The cryptographic building blocks used in our FSC construction are precisely the following:

- $\mathcal{O}$: An indistinguishability obfuscator for P/poly.

- PKE=(PKE.KeyGen, PKE.Encrypt, PKE.Decrypt): A CPA-secure public key encryption scheme with message space $\mathbb{M} \subseteq \{0,1\}^{n(\lambda)}$, for some polynomial n.

– SIG=($\mathsf{SIG.KeyGen}$, $\mathsf{SIG.Sign}$, $\mathsf{SIG.Verify}$): An existentially unforgeable signature scheme with message space $\{0,1\}^\lambda$.

– $\mathsf{SSS\text{-}NIZKPoK}$ = ($\mathsf{SSS\text{-}NIZKPoK.Setup}$, $\mathsf{SSS\text{-}NIZKPoK.Prove}$, $\mathsf{SSS\text{-}NIZKPoK.Verify}$, $\mathsf{SSS\text{-}NIZKPoK.SimSetup}$, $\mathsf{SSS\text{-}NIZKPoK.SimProve}$, $\mathsf{SSS\text{-}NIZKPoK.ExtSetup}$, $\mathsf{SSS\text{-}NIZKPoK.Extr}$): An $\mathsf{SSS\text{-}NIZKPoK}$ system for the NP relation R, whose statements are of the form $X = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1, e_2) \in \{0,1\}^*$, witnesses are of the form $W = (m, r_1, r_2, f, \sigma, z) \in \{0,1\}^*$, and

$$(X, W) \in R \iff (e_1 = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(1)}_{\mathsf{PKE}}, m; r_1) \bigwedge e_2 = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(2)}_{\mathsf{PKE}}, m; r_2) \bigwedge \mathsf{SIG.Verify}(\mathrm{VK}_{\mathsf{SIG}}, f, \sigma) = 1 \bigwedge m = f(z)), \quad (2)$$

for a function family $\mathbb{F} = \{f : \mathbb{D}_f \to \mathbb{M}\} \subseteq \mathsf{P/poly}$ (with representation in $\{0,1\}^\lambda$).

We build an FSC scheme for message space $\mathbb{M}$, family of signing functions $\mathbb{F}$, and the class of decryption functions $\mathbb{G} = \{g : \mathbb{M} \to \mathbb{R}_g\} \subseteq \mathsf{P/poly}$.

4.1 Construction

$\mathsf{FSC.Setup}(1^\lambda)$: The trusted authority takes in a security parameter 1^λ and proceeds as follows:
1. It generates$(\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{SK}^{(1)}_{\mathsf{PKE}}), (\mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{SK}^{(2)}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$.
2. It obtains $(\mathrm{VK}_{\mathsf{SIG}}, \mathrm{SK}_{\mathsf{SIG}}) \leftarrow \mathsf{SIG.KeyGen}(1^\lambda)$.
3. It generates $\mathrm{CRS} \leftarrow \mathsf{SSS\text{-}NIZKPoK.Setup}(1^\lambda)$.
4. It publishes the public parameters $\mathrm{MPK} = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, \mathrm{CRS})$, while keeps the master secret key $\mathrm{MSK} = (\mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{SK}_{\mathsf{SIG}})$ to itself.

$\mathsf{FSC.SKeyGen}(\mathrm{MPK}, \mathrm{MSK}, f)$: Taking as input the public parameters MPK, the master secret key MSK, and a signing function $f \in \mathbb{F}$ from a signcrypter, the trusted authority runs $\mathsf{SIG.Sign}(\mathrm{SK}_{\mathsf{SIG}}, f)$ to obtain a signature σ on f and return the signing key $\mathrm{SK}(f) = (f, \sigma)$ to the signcrypter.

$\mathsf{FSC.Signcrypt}(\mathrm{MPK}, \mathrm{SK}(f), z)$: A signcrypter takes as input the public parameters MPK, its signing key $\mathrm{SK}(f) = (f, \sigma)$ corresponding to some signing function $f \in \mathbb{F}$, and an input $z \in \mathbb{D}_f$. It prepares the ciphertext as follows:
1. It computes $e_\ell = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(\ell)}_{\mathsf{PKE}}, f(z);\ r_\ell)$ for $\ell = 1, 2$, where r_ℓ is the randomness selected for encryption.
2. It generates a proof $\pi \leftarrow \mathsf{SSS\text{-}NIZKPoK.Prove}(\mathrm{CRS}, X, W)$ where $X = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1, e_2)$ is a statement of the NP relation R defined in equation (2) and $W = (f(z), r_1, r_2, f, \sigma, z)$ is the corresponding witness.
3. It outputs the ciphertext $\mathrm{CT} = (e_1, e_2, \pi)$.

$\mathsf{FSC.DKeyGen}(\mathrm{MPK}, \mathrm{MSK}, g)$: On input the public parameters MPK, the master secret key MSK, and a decryption function $g \in \mathbb{G}$ from a decrypter, the trusted authority computes the obfuscation $\mathcal{O}(P^{(g, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})})$ of the program $P^{(g, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})}$ using the circuit size value equal to $\max\{|P^{(g, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})}|, |\widetilde{P}^{(g, \mathrm{SK}^{(2)}_{\mathsf{PKE}}, \mathrm{MPK})}|\}$, where the programs $P^{(g, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})}$ and $\widetilde{P}^{(g, \mathrm{SK}^{(2)}_{\mathsf{PKE}}, \mathrm{MPK})}$ are described in Figure 1. It provides the decryption key $\mathrm{DK}(g) = (g, \mathcal{O}(P^{(g, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})}))$ to the decrypter.

$\mathsf{FSC.Unsigncrypt}(\mathrm{MPK}, \mathrm{DK}(g), \mathrm{CT})$: A decrypter, on input the public parameters MPK, its decryption key $\mathrm{DK}(g) = (g, \mathcal{O}(P^{(g, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})}))$, along with a ciphertext $\mathrm{CT} = (e_1, e_2, \pi)$, runs the obfuscated program $\mathcal{O}(P^{(g, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})})$ with input (e_1, e_2, π) and outputs the result.

$P^{(g,\mathrm{SK}^{(1)}_{\mathsf{PKE}},\mathrm{MPK})}$	$\widetilde{P}^{(g,\mathrm{SK}^{(2)}_{\mathsf{PKE}},\mathrm{MPK})}$
Given input (e_1, e_2, π), the program proceeds as follows:	Given input (e_1, e_2, π), the program proceeds as follows:
1. Extract $\mathrm{PK}^{(1)}_{\mathsf{PKE}}$, $\mathrm{PK}^{(2)}_{\mathsf{PKE}}$, $\mathrm{VK}_{\mathsf{SIG}}$, CRS from MPK. 2. Set $X = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1, e_2)$. 3. If $\mathsf{SSS\text{-}NIZKPoK.Verify}(\mathrm{CRS}, X, \pi) = 0$, then output $\perp$ and stop. Otherwise, continue to the next step. 4. Output $g\big(\mathsf{PKE.Decrypt}(\mathrm{SK}^{(1)}_{\mathsf{PKE}}, e_1)\big)$.	1. Extract $\mathrm{PK}^{(1)}_{\mathsf{PKE}}$, $\mathrm{PK}^{(2)}_{\mathsf{PKE}}$, $\mathrm{VK}_{\mathsf{SIG}}$, CRS from MPK. 2. Set $X = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1, e_2)$. 3. If $\mathsf{SSS\text{-}NIZKPoK.Verify}(\mathrm{CRS}, X, \pi) = 0$, then output $\perp$ and stop. Otherwise, continue to the next step. 4. Output $g\big(\mathsf{PKE.Decrypt}(\mathrm{SK}^{(2)}_{\mathsf{PKE}}, e_2)\big)$.

Fig. 1

Correctness: Note that the correctness of the proposed scheme follows immediately from the correctness of $\mathcal{O}$, PKE, and SIG, perfect completeness of SSS-NIZKPoK systems, as well as description of the program template $P^{(g,\mathrm{SK}^{(1)}_{\mathsf{PKE}},\mathrm{MPK})}$.

Remark 1. Note that the size of the ciphertext in our FSC scheme is $\tau(\lambda, n)$ for some polynomial τ.

4.2 Security Analysis

Theorem 1 (Message Confidentiality of FSC). *Assuming* IO $\mathcal{O}$ *for* P/poly, CPA*-secure public key encryption* PKE*, along with the statistical simulation-soundness and zero-knowledge properties of* SSS-NIZKPoK *system, the* FSC *scheme described in §4.1 is selectively message confidential against* CPA *as per the definition given in §3.*

Proof. Suppose that any adversary in the selective CPA-message confidentiality game of §3 makes at most $q = q(\lambda)$ many decryption key queries. For simplicity, we assume that the adversary always makes exactly q decryption key queries. We denote g_i for $i \in [q]$ to be the i-th decryption function for which a decryption key query is made. By the rules of the game $g_i(f^*_0(z^*_0))$ is constrained to be equal to $g_i(f^*_1(z^*_1))$ for all $i \in [q]$.

We organize our proof into a sequence of hybrids. In the first hybrid the challenger signcrypts $f^*_0(z^*_0)$. We then gradually change the signcryption in multiple hybrid steps into a signcryption of $f^*_1(z^*_1)$ in the challenge ciphertext. We show that each hybrid experiment is indistinguishable from the previous one, thus showing our FSC scheme to have selective message confidentiality against CPA.

Sequence of Hybrids:

- $\mathsf{Hyb_0}$: This corresponds to the honest execution of the selective CPA-message confidentiality game introduced in §3 when the challenger signcrypts $f^*_0(z^*_0)$ in the challenge ciphertext $\mathrm{CT}^* = (e^*_1, e^*_2, \pi^*)$, i.e., $e^*_\ell = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(\ell)}_{\mathsf{PKE}}, f^*_0(z^*_0); r^*_\ell)$ for $\ell = 1, 2$ and $\pi^* \leftarrow \mathsf{SSS\text{-}NIZKPoK.Prove}(\mathrm{CRS}, X^*, W^*)$ where $X^* = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e^*_1, e^*_2)$ and W^* is a valid witness corresponding to X^*.

- $\mathsf{Hyb_1}$: In this hybrid, the common reference string CRS included in the public parameters MPK is generated as $(\mathrm{CRS}, \mathrm{TR}) \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimSetup}(1^\lambda, X^*)$, and the proof π^* included in the challenge ciphertext CT^* is simulated as $\pi^* \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimProve}(\mathrm{CRS}, \mathrm{TR}, X^*)$ where $X^* = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e^*_1, e^*_2)$. The rest of the experiment continues as in Hyb_0 using the simulated common reference string CRS.

- $\mathsf{Hyb_2}$: This hybrid is the same as the last hybrid except that the challenge ciphertext is computed as $\mathrm{CT}^* = (e^*_1 = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(1)}_{\mathsf{PKE}}, f^*_0(z^*_0); r^*_1), e^*_2 = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(2)}_{\mathsf{PKE}}, f^*_1(z^*_1); r^*_2),$

$\pi^* \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimProve}(\mathrm{CRS}, \mathrm{TR}, X^*))$ where $X^* = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e^*_1, e^*_2)$.

- $\mathsf{Hyb}_{\mathbf{3},i}$ for $i \in [0, q]$: In this sequence of hybrids, we change the form of the decryption keys provided to the adversary in response to its decryption key queries. In $\mathsf{Hyb}_{3,i}$, for $i \in [0, q]$, the first i decryption keys requested by the adversary will result in decryption keys generated as $\mathrm{DK}(g_i) = \big(g_i, \mathcal{O}(\widetilde{P}^{(g_i, \mathrm{SK}^{(2)}_{\mathsf{PKE}}, \mathrm{MPK})})\big)$ while the remaining $i+1$ to q decryption keys are generated as $\mathrm{DK}(g_i) = \big(g_i, \mathcal{O}(P^{(g_i, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})})\big)$ as in Hyb_2, where $P^{(g_i, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})}$ and $\widetilde{P}^{(g_i, \mathrm{SK}^{(2)}_{\mathsf{PKE}}, \mathrm{MPK})}$ are depicted in Figure 1. Observe that $\mathsf{Hyb}_{3,0}$ is equivalent to Hyb_2.

- $\mathsf{Hyb}_{\mathbf{4}}$: This hybrid is identical to the hybrid $\mathsf{Hyb}_{3,q}$ with the exception that the challenge ciphertext is generated as $\mathrm{CT}^* = (e^*_1, e^*_2, \pi^*)$ where $e^*_1 = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(1)}_{\mathsf{PKE}}, f^*_1(z^*_1); r^*_1), e^*_2 = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(2)}_{\mathsf{PKE}}, f^*_1(z^*_1); r^*_2)$, and the proof π^* is still simulated.

- $\mathsf{Hyb}_{\mathbf{5},i}$ for $i \in [0, q]$: In this sequence of hybrids, we again change the form of the decryption keys returned to the adversary in response to its decryption key queries. In $\mathsf{Hyb}_{5,i}$, for $i \in [0, q]$, the first i decryption key queries of the adversary will result in decryption keys generated as $\mathrm{DK}(g_i) = \big(g_i, \mathcal{O}(P^{(g_i, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})})\big)$ while the rest of the decryption keys $i+1$ to q are generated as $\mathrm{DK}(g_i) = \big(g_i, \mathcal{O}(\widetilde{P}^{(g_i, \mathrm{SK}^{(2)}_{\mathsf{PKE}}, \mathrm{MPK})})\big)$ as in Hyb_4, where $P^{(g_i, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})}$ and $\widetilde{P}^{(g_i, \mathrm{SK}^{(2)}_{\mathsf{PKE}}, \mathrm{MPK})}$ are defined in Figure 1. Note that $\mathsf{Hyb}_{5,0}$ is equivalent to Hyb_4.

- $\mathsf{Hyb}_{\mathbf{6}}$: In this hybrid, the common reference string CRS included in MPK is obtained as $\mathrm{CRS} \leftarrow \mathsf{SSS\text{-}NIZKPoK.Setup}(1^\lambda)$ and the proof π^* included in the challenge ciphertext CT^* is generated as $\pi^* \leftarrow \mathsf{SSS\text{-}NIZKPoK.Prove}(\mathrm{CRS}, X^*, W^*)$ where $X^* = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e^*_1, e^*_2)$ and W^* is a valid witness corresponding to X^*. The remainder of the experiment continues identically as in $\mathsf{Hyb}_{5,q}$ using the honestly generated common reference string CRS. Notice that this hybrid corresponds to the selective CPA-message confidentiality game when $f^*_1(z^*_1)$ is signcrypted in the challenge ciphertext.

Proofs of Hybrid Arguments:

We will present a sequence of lemmas which will demonstrate that no PPT adversary can distinguish with non-negligible advantage between any two consecutive hybrids described above, and thus security in the selective CPA-message confidentiality game follows.

Lemma 1. *Assuming* SSS-NIZKPoK *system is computationally zero-knowledge, no* PPT *adversary can distinguish with non-negligible advantage between* Hyb_0 *and* Hyb_1.

Proof. Suppose there is a PPT adversary $\mathcal{A}$ that can distinguish with non-negligible advantage between Hyb_0 and Hyb_1. We construct a PPT algorithm $\mathcal{C}$ that breaks the zero-knowledge property of SSS-NIZKPoK using $\mathcal{A}$ as a subroutine. $\mathcal{C}$ interacts with $\mathcal{A}$ as follows:

- $\mathcal{C}$ begins by initializing $\mathcal{A}$ and receiving $(f^*_0, z^*_0), (f^*_1, z^*_1)$ from $\mathcal{A}$.
- In order to setup the public parameters, $\mathcal{C}$ proceeds as follows:
  - $\mathcal{C}$ itself generates $(\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{SK}^{(1)}_{\mathsf{PKE}}), (\mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{SK}^{(2)}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and $(\mathrm{VK}_{\mathsf{SIG}}, \mathrm{SK}_{\mathsf{SIG}}) \leftarrow \mathsf{SIG.KeyGen}(1^\lambda)$.
  - After that, it computes $e^*_\ell = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(\ell)}_{\mathsf{PKE}}, f^*_0(z^*_0); r^*_\ell)$ using randomness r^*_ℓ, for $\ell = 1, 2$ and $\sigma^* \leftarrow \mathsf{SIG.Sign}(\mathrm{SK}_{\mathsf{SIG}}, f^*_0)$.

– It then submits the statement $X^* = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e^*_1, e^*_2)$ along with the corresponding witness $W^* = (f^*_0(z^*_0), r^*_1, r^*_2, f^*_0, \sigma^*, z^*_0)$ to its *zero-knowledge challenger* $\mathcal{B}$ and receives back a common reference string CRS' together with a proof π'^* on X^* from $\mathcal{B}$.

– $\mathcal{C}$ hands the public parameters $\mathrm{MPK} = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, \mathrm{CRS} = \mathrm{CRS}')$ to $\mathcal{A}$ and keeps $\widetilde{\mathrm{MSK}} = (\mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{SK}_{\mathsf{SIG}}, e^*_1, e^*_2, \pi'^*)$.

- The signing key, decryption key, and signcryption queries of $\mathcal{A}$ are answered by $\mathcal{C}$ as described below:

  – *Signing key query*: Since $\mathcal{C}$ knows $\mathrm{SK}_{\mathsf{SIG}}$, it answers any signing key query of $\mathcal{A}$ for any signing function $f \in \mathbb{F}$ by generating $\sigma \leftarrow \mathsf{SIG.Sign}(\mathrm{SK}_{\mathsf{SIG}}, f)$ and returning $\mathrm{SK}(f) = (f, \sigma)$ to $\mathcal{A}$.

  – *Decryption key query*: Using $\mathrm{SK}^{(1)}_{\mathsf{PKE}}$ and MPK, $\mathcal{C}$ constructs the program $P^{(g_i, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})}$ described in Figure 1 upon receiving a decryption key query from $\mathcal{A}$ corresponding to a decryption function $g_i \in \mathbb{G}$, and provides the decryption key $\mathrm{DK}(g_i) = \big(g_i, \mathcal{O}(P^{(g_i, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})})\big)$ to $\mathcal{A}$.

  – *Signcryption query*: When $\mathcal{A}$ makes a signcryption query corresponding to a signing function $f \in \mathbb{F}$ and input $z \in \mathbb{D}_f$, $\mathcal{C}$ first computes $\sigma \leftarrow \mathsf{SIG.Sign}(\mathrm{SK}_{\mathsf{SIG}}, f)$, $e_\ell = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(\ell)}_{\mathsf{PKE}}, f(z); r_\ell)$ using randomness r_ℓ, for $\ell = 1, 2$, along with a proof $\pi \leftarrow \mathsf{SSS\text{-}NIZKPoK.Prove}(\mathrm{CRS}, X, W)$ where $X = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1, e_2)$ and $W = (f(z), r_1, r_2, f, \sigma, z)$. It provides the ciphertext $\mathrm{CT} = (e_1, e_2, \pi)$ to $\mathcal{A}$.

- $\mathcal{C}$ sends the challenge ciphertext $\mathrm{CT}^* = (e^*_1, e^*_2, \pi^* = \pi'^*)$ to $\mathcal{A}$.

- Finally, $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$. $\mathcal{C}$ also outputs b'.

Note that if $\mathcal{B}$ used the real setup algorithm $\mathsf{SSS\text{-}NIZKPoK.Setup}(1^\lambda)$ to generate CRS' and real prover $\mathsf{SSS\text{-}NIZKPoK.Prove}(\mathrm{CRS}', X^*, W^*)$ to generate the proof π'^*, then we are exactly in Hyb_0. On the other hand, if the common reference string and the proof are simulated, then we are in Hyb_1. Thus, if $\mathsf{View}_{\mathsf{Hyb}_0}$ and $\mathsf{View}_{\mathsf{Hyb}_1}$ respectively denote the views of $\mathcal{A}$ in the hybrids Hyb_0 and Hyb_1, then we have

$$
\begin{aligned}
&\big|\mathsf{Pr}\big[\mathcal{A}(\mathsf{View}_{\mathsf{Hyb}_0}) = 1\big] - \mathsf{Pr}\big[\mathcal{A}(\mathsf{View}_{\mathsf{Hyb}_1}) = 1\big]\big| \\
&= \big|\mathsf{Pr}\big[\mathrm{CRS}' \leftarrow \mathsf{SSS\text{-}NIZKPoK.Setup}(1^\lambda); \pi'^* \leftarrow \mathsf{SSS\text{-}NIZKPoK.Prove}(\mathrm{CRS}', X^*, W^*) : \\
&\quad \mathcal{C}(\mathrm{CRS}', X^*, \pi'^*) = 1\big] \;-\; \mathsf{Pr}\big[(\mathrm{CRS}', \mathrm{TR}) \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimSetup}(1^\lambda, X^*); \\
&\quad \pi'^* \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimProve}(\mathrm{CRS}', \mathrm{TR}, X^*) : \mathcal{C}(\mathrm{CRS}', X^*, \pi'^*) = 1\big]\big| = \mathsf{Adv}^{\mathsf{SSS\text{-}NIZKPoK,ZK}}_{\mathcal{C}}(\lambda).
\end{aligned}
$$

Hence the lemma. □

Lemma 2. *Assuming* PKE *is* CPA *secure, no* PPT *adversary can distinguish with non-negligible advantage between the hybrids* Hyb_1 *and* Hyb_2.

Proof. Suppose there is a PPT adversary $\mathcal{A}$ that can distinguish with non-negligible advantage between Hyb_1 and Hyb_2. We construct a PPT algorithm $\mathcal{C}$ that breaks the CPA-security of PKE using $\mathcal{A}$ as a sub-routine. $\mathcal{C}$ interacts with $\mathcal{A}$ as follows:

- $\mathcal{C}$ begins by initializing $\mathcal{A}$ and receiving $(f^*_0, z^*_0), (f^*_1, z^*_1)$ from $\mathcal{A}$.

- To setup the public parameters, $\mathcal{C}$ proceeds as follows:
  - $\mathcal{C}$ itself generates $(\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{SK}^{(1)}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$.
  - It also receives a public key $\mathrm{PK}'_{\mathsf{PKE}}$ for PKE from its *CPA-security challenger* $\mathcal{B}$ and sets $\mathrm{PK}^{(2)}_{\mathsf{PKE}} = \mathrm{PK}'_{\mathsf{PKE}}$.
  - Then, it computes $e^*_1 = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(1)}_{\mathsf{PKE}}, f^*_0(z^*_0); r^*_1)$ itself using randomness r^*_1.
  - Next, it sends the two messages $f^*_0(z^*_0), f^*_1(z^*_1)$ to $\mathcal{B}$ which sends back a challenge ciphertext e'^*. It designates $e^*_2 = e'^*$.
  - After that, $\mathcal{C}$ itself generates $(\mathrm{VK}_{\mathsf{SIG}}, \mathrm{SK}_{\mathsf{SIG}}) \leftarrow \mathsf{SIG.KeyGen}(1^\lambda)$ together with $(\mathrm{CRS}, \mathrm{TR}) \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimSetup}(1^\lambda, X^*)$ where $X^* = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e^*_1, e^*_2)$.
  - It hands the public parameters $\mathrm{MPK} = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, \mathrm{CRS})$ to $\mathcal{A}$ while keeps $\widetilde{\mathrm{MSK}} = (\mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{SK}_{\mathsf{SIG}}, \mathrm{TR}, e^*_1, e^*_2)$.
- Using $\mathrm{SK}_{\mathsf{SIG}}$ and $\mathrm{SK}^{(1)}_{\mathsf{PKE}}$, the signing key, decryption key, and signcryption queries of $\mathcal{A}$ are answered by $\mathcal{C}$ in an analogous fashion as in the proof of Lemma 1.
- To compute the challenge ciphertext, $\mathcal{C}$ computes $\pi^* \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimProve}(\mathrm{CRS}, \mathrm{TR}, X^*)$ where $X^* = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e^*_1, e^*_2)$. It sends the challenge ciphertext $\mathrm{CT}^* = (e^*_1, e^*_2, \pi^*)$ to $\mathcal{A}$.
- Eventually, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. $\mathcal{C}$ also outputs b'.

Observe that, if $\mathcal{B}$ gave $e'^* \leftarrow \mathsf{PKE.Encrypt}(\mathrm{PK}'_{\mathsf{PKE}}, f^*_0(z^*_0))$, then we are exactly in hybrid Hyb_1. On the other hand, if it gave $e'^* \leftarrow \mathsf{PKE.Encrypt}(\mathrm{PK}'_{\mathsf{PKE}}, f^*_1(z^*_1))$, then we are in Hyb_2. Thus, if we denote by $\mathsf{View}_{\mathsf{Hyb}_1}$ and $\mathsf{View}_{\mathsf{Hyb}_2}$ the views of $\mathcal{A}$ in the hybrids Hyb_1 and Hyb_2 respectively, then we have

$$
\begin{aligned}
&\big|\mathsf{Pr}\big[\mathcal{A}(\mathsf{View}_{\mathsf{Hyb}_1}) = 1\big] - \mathsf{Pr}\big[\mathcal{A}(\mathsf{View}_{\mathsf{Hyb}_2}) = 1\big]\big| \\
&= \big|1 - \mathsf{Pr}\big[(\mathrm{PK}'_{\mathsf{PKE}}, \mathrm{SK}'_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda);\ (f^*_0(z^*_0), f^*_1(z^*_1)) \leftarrow \mathcal{C}(\mathrm{PK}'_{\mathsf{PKE}}); \\
&\qquad\qquad e'^* \leftarrow \mathsf{PKE.Encrypt}(\mathrm{PK}'_{\mathsf{PKE}}, f^*_0(z^*_0))\ :\ \mathcal{C}(\mathrm{PK}'_{\mathsf{PKE}}, e'^*) = 0\big] \\
&\quad - \mathsf{Pr}\big[(\mathrm{PK}'_{\mathsf{PKE}}, \mathrm{SK}'_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda);\ (f^*_0(z^*_0), f^*_1(z^*_1)) \leftarrow \mathcal{C}(\mathrm{PK}'_{\mathsf{PKE}}); \\
&\qquad\qquad e'^* \leftarrow \mathsf{PKE.Encrypt}(\mathrm{PK}'_{\mathsf{PKE}}, f^*_1(z^*_1))\ :\ \mathcal{C}(\mathrm{PK}'_{\mathsf{PKE}}, e'^*) = 1\big]\big| = 2\mathsf{Adv}^{\mathsf{PKE,IND\text{-}CPA}}_{\mathcal{C}}(\lambda),
\end{aligned}
$$

where $\mathsf{Adv}^{\mathsf{PKE,IND\text{-}CPA}}_{\mathcal{C}}(\lambda)$ denotes the advantage of $\mathcal{C}$ in the CPA-security game for PKE. Hence the lemma. □

Lemma 3. *Assuming $\mathcal{O}$ is an* IO *for* P/poly *and* SSS-NIZKPoK *is statistically simulation-sound, no* PPT *adversary can distinguish with non-negligible advantage between $\mathsf{Hyb}_{3,i}$ and $\mathsf{Hyb}_{3,i+1}$ for $i \in [0, q-1]$.*

Proof. Suppose that there is a PPT adversary $\mathcal{A}$ that can distinguish with non-negligible advantage between $\mathsf{Hyb}_{3,i}$ and $\mathsf{Hyb}_{3,i+1}$. We build a PPT algorithm $\mathcal{C}$ that breaks the IO property of $\mathcal{O}$ using $\mathcal{A}$ as a subroutine. $\mathcal{C}$ interacts with $\mathcal{A}$ as follows:

- $\mathcal{C}$ starts with initializing $\mathcal{A}$ and obtaining $(f^*_0, z^*_0), (f^*_1, z^*_1)$ from $\mathcal{A}$.

- In order to setup the public parameters, $\mathcal{C}$ performs the following steps:
  - $\mathcal{C}$ first generates $(\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{SK}^{(1)}_{\mathsf{PKE}}), (\mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{SK}^{(2)}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and $(\mathrm{VK}_{\mathsf{SIG}}, \mathrm{SK}_{\mathsf{SIG}}) \leftarrow \mathsf{SIG.KeyGen}(1^\lambda)$.
  - It computes $e^*_1 = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(1)}_{\mathsf{PKE}}, f^*_0(z^*_0); r^*_1)$ and $e^*_2 = \mathsf{PKE.Encrypt}(\mathrm{PK}^{(2)}_{\mathsf{PKE}}, f^*_1(z^*_1); r^*_2)$ using randomness r^*_1 and r^*_2 respectively.
  - Then it obtains $(\mathrm{CRS}, \mathrm{TR}) \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimSetup}(1^\lambda, X^*)$ where $X^* = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e^*_1, e^*_2)$.
  - It gives the public parameters $\mathrm{MPK} = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, \mathrm{CRS})$ to $\mathcal{A}$ while keeps $\widetilde{\mathrm{MSK}} = (\mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{SK}^{(2)}_{\mathsf{PKE}}, \mathrm{SK}_{\mathsf{SIG}}, \mathrm{TR}, e^*_1, e^*_2)$ to itself.
- The signing key and signcryption key queries of $\mathcal{A}$ are answered by $\mathcal{C}$ in the same manner as in the proof of Lemma 2 using $\mathrm{SK}_{\mathsf{SIG}}$. Now consider the decryption key queries made by $\mathcal{A}$. Recall that $\mathcal{A}$ makes q decryption key queries corresponding to decryption functions $g_i \in \mathbb{G}$. The answers to these queries are provided as follows:
  (a) For $j \leq i$, $\mathcal{C}$ creates the j-th decryption key $\mathrm{DK}(g_j) = \big(g_j, \mathcal{O}(\widetilde{P}^{(g_j, \mathrm{SK}^{(2)}_{\mathsf{PKE}}, \mathrm{MPK})})\big)$. Note that $\mathcal{C}$ knows $\mathrm{SK}^{(2)}_{\mathsf{PKE}}$ and therefore can form the program $\widetilde{P}^{(g_j, \mathrm{SK}^{(2)}_{\mathsf{PKE}}, \mathrm{MPK})}$ itself.
  (b) For $j > i{+}1$, the j-th queried decryption key is created as $\mathrm{DK}(g_j) = \big(g_j, \mathcal{O}(P^{(g_j, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})})\big)$ by $\mathcal{C}$ using $\mathrm{SK}^{(1)}_{\mathsf{PKE}}$.
  (c) For the $(i+1)$-th decryption key query, $\mathcal{C}$ submits $C_0 = P^{(g_{i+1}, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})}$ and $C_1 = \widetilde{P}^{(g_{i+1}, \mathrm{SK}^{(2)}_{\mathsf{PKE}}, \mathrm{MPK})}$ to its IO *challenger* $\mathcal{B}$ and receives back an obfuscated circuit C'. $\mathcal{C}$ gives $\mathrm{DK}(g_{i+1}) = (g_{i+1}, C')$ to $\mathcal{A}$.
- To prepare the challenge ciphertext, $\mathcal{C}$ computes $\pi^* \leftarrow \mathsf{SSS\text{-}NIZKPoK.SimProve}(\mathrm{CRS}, \mathrm{TR}, X^*)$ where $X^* = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e^*_1, e^*_2)$. $\mathcal{C}$ sends the challenge ciphertext $\mathrm{CT}^* = (e^*_1, e^*_2, \pi^*)$ to $\mathcal{A}$.
- Eventually, $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$. $\mathcal{C}$ also outputs b'.

We now argue that (C_0, C_1) forms a valid instance of the IO assumption by exhibiting that both the programs $P^{(g_{i+1}, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})}$ and $\widetilde{P}^{(g_{i+1}, \mathrm{SK}^{(2)}_{\mathsf{PKE}}, \mathrm{MPK})}$ described in Figure 1 produce the same output on each input. We break our argument into cases on the input to these programs.

(I) We first consider inputs (e_1, e_2, π) where e_1, e_2 are valid encryption of the same message and π is a proof of the statement $X = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1, e_2)$ for which $\mathsf{SSS\text{-}NIZKPoK.Verify}(\mathrm{CRS}, X, \pi) = 1$. For these inputs both programs will reach Step 4 where they decrypt the same message, no matter whether they use $\mathrm{SK}^{(1)}_{\mathsf{PKE}}$ or $\mathrm{SK}^{(2)}_{\mathsf{PKE}}$, and compute the same function g_{i+1} on the decrypted message. Thus the output of both programs is the same on all inputs of this class.

(II) The next set of inputs considered are (e_1, e_2, π) for which $\mathsf{SSS\text{-}NIZKPoK.Verify}\big(\mathrm{CRS}, (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1, e_2), \pi\big)$ in Step 3 of both programs outputs 0. In this case both the programs output $\perp$.

(III) Finally, we consider the set of inputs (e_1, e_2, π) for which $\mathsf{SSS\text{-}NIZKPoK.Verify}\big(\mathrm{CRS}, (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1, e_2), \pi\big)$ in Step 3 of both the programs outputs 1 but e_1, e_2 are not valid encryptions of the same message. Note that, due to the statistical simulation-soundness property of $\mathsf{SSS\text{-}NIZKPoK}$, with all but negligible probability this can happen only when

$e_1 = e_1^*$ and $e_2 = e_2^*$, and hence, decrypting e_1 gives $f_0^*(z_0^*)$ while decrypting e_2 results in $f_1^*(z_1^*)$. However, $P^{(g_{i+1},\mathrm{SK}^{(1)}_{\mathsf{PKE}},\mathrm{MPK})}$ outputs $g_{i+1}(f_0^*(z_0^*))$ which is bound to be equal to $g_{i+1}(f_1^*(z_1^*))$ (by the rules of the game), which is the output of $\widetilde{P}^{(g_{i+1},\mathrm{SK}^{(2)}_{\mathsf{PKE}},\mathrm{MPK})}$. Thus, we can see that both programs have the same output for this input class as well.

Now, observe that, if $\mathcal{B}$ gave $C' = \mathcal{O}(C_0)$, then we are exactly in $\mathsf{Hyb}_{3,i}$. Whereas, if it gave $C' = \mathcal{O}(C_1)$, then we are in $\mathsf{Hyb}_{3,i+1}$. Hence, we can derive a relation similar to that in the proof of Lemma 2 between the advantages of $\mathcal{A}$ and $\mathcal{C}$. The lemma follows. □

Lemma 4. *Assuming* PKE *is* CPA *secure, no* PPT *adversary can distinguish with non-negligible advantage between* $\mathsf{Hyb}_{3,q}$ *and* Hyb_4.

Lemma 5. *Assuming* $\mathcal{O}$ *is an* IO *for* P/poly *and* SSS-NIZKPoK *is statistically simulation-sound, no* PPT *adversary can distinguish with non-negligible advantage between* $\mathsf{Hyb}_{5,i}$ *and* $\mathsf{Hyb}_{5,i+1}$ *for* $i \in [0, q]$.

Lemma 6. *Assuming* SSS-NIZKPoK *is computationally zero-knowledge, no* PPT *adversary can distinguish with non-negligible advantage between* $\mathsf{Hyb}_{5,q}$ *and* Hyb_6.

The proofs of Lemmas 4, 5, and 6 follow in a directly analogous manner to those of Lemmas 2, 3, and 1 respectively. □

Theorem 2 (Ciphertext Unforgeability of FSC). *Under the assumption that* SIG *is existentially unforgeable against* CMA *and* SSS-NIZKPoK *is a proof of knowledge, the* FSC *construction of §4.1 is selectively ciphertext unforgeable against* CMA *as per the definition given in §3.*

Proof. Here also we will organize the proof in two hybrid experiments. The first hybrid corresponds to the real execution of the selective ciphertext unforgeability game of §3. In the second hybrid, we generate the common reference string included in the public parameters provided to the adversary using the extraction-enabling setup procedure and run the extraction process on the forgery provided by the adversary. Note that by the rules of the game, all the *signing key queries* of the adversary are restricted to functions $f \in \mathbb{F}$ for which there does not exist any $z \in \mathbb{D}_f$ with $f(z) = m^*$ and all the *signcryption queries* of the adversary is constrained to pairs (f, z) such that $f(z) \neq m^*$, where $m^* \in \mathbb{M}$ is the message committed by the adversary in the initialization phase. We prove that the advantage of any PPT adversary in the first hybrid, i.e., in the real selective ciphertext unforgeability game is negligibly different from that in the second hybrid, and moreover, the advantage of any PPT adversary in the second hybrid experiment itself is negligible. The theorem will then follow immediately.

Sequence of Hybrids:

- $\mathsf{Hyb}_\mathbf{0}$: This hybrid corresponds to the honest execution of the selective CMA-ciphertext unforgeability game defined in §3. Let Forge_0 denotes the advantage of any PPT adversary in this experiment. Thus, in other words, the advantage of any PPT adversary in the selective CMA-ciphertext unforgeability game is equal to Forge_0.

- $\mathsf{Hyb}_\mathbf{1}$: This hybrid is the same as the previous one with the exception that the common reference string CRS included in MPK provided to the adversary is generated as $(\mathrm{CRS}, \widehat{\mathrm{TR}}) \leftarrow \mathsf{SSS\text{-}NIZKPoK.ExtSetup}(1^\lambda)$. The remainder of the experiment continues as before with respect to CRS. At the end of the experiment, the extraction algorithm is executed on the adversary's alleged forgery $\mathrm{CT}^* = (e_1^*, e_2^*, \pi^*)$ to extract the witness W^*, i.e., to be precise, $W^* \leftarrow \mathsf{SSS\text{-}NIZKPoK.Extr}(\mathrm{CRS}, \widehat{\mathrm{TR}}, X^*, \pi^*)$ is performed, where $X^* = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1^*, e_2^*)$. Let Forge_1 denotes the advantage of any PPT adversary in this hybrid.

Proofs of Hybrid Arguments:

Lemma 7. *Assuming that the common reference string generated by* SSS-NIZKPoK.Setup *and that produced by* SSS-NIZKPoK.ExtSetup *are statistically close, the difference between* Forge_0 *and* Forge_1 *is negligible for all* PPT *adversaries.*

Proof. Consider a PPT adversary $\mathcal{A}$ for which the difference between Forge_0 and Forge_1 is non-negligible. We construct a probabilistic (not necessarily polynomial-time) algorithm $\mathcal{C}$ that distinguishes between common reference string generated by SSS-NIZKPoK.Setup and that obtained from SSS-NIZKPoK.ExtSetup using $\mathcal{A}$ as a subroutine. $\mathcal{C}$ interacts with $\mathcal{A}$ as follows:

- $\mathcal{C}$ starts by initializing $\mathcal{A}$ and receiving m^* from $\mathcal{A}$.
- In order to setup the public parameters, $\mathcal{C}$ proceeds as follows:
  - $\mathcal{C}$ generates $(\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{SK}^{(1)}_{\mathsf{PKE}}), (\mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{SK}^{(2)}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and $(\mathrm{VK}_{\mathsf{SIG}}, \mathrm{SK}_{\mathsf{SIG}}) \leftarrow \mathsf{SIG.KeyGen}(1^\lambda)$.
  - $\mathcal{C}$ also receives a common reference string CRS' from its *common reference string challenger* $\mathcal{B}$ and sets $\mathrm{CRS} = \mathrm{CRS}'$.
  - $\mathcal{C}$ hands the public parameters $\mathrm{MPK} = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, \mathrm{CRS})$ to $\mathcal{A}$ while keeps $\widetilde{\mathrm{MSK}} = (\mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{SK}^{(2)}_{\mathsf{PKE}}, \mathrm{SK}_{\mathsf{SIG}})$.
- The signing key, decryption key, signcryption, and unsigncryption queries of $\mathcal{A}$ are answered by $\mathcal{C}$ as described below:
  - *Signing key query*: Note that $\mathcal{C}$ knows $\mathrm{SK}_{\mathsf{SIG}}$ and hence it can answer a signing key query of $\mathcal{A}$ for a signing function $f \in \mathbb{F}$ by generating $\sigma \leftarrow \mathsf{SIG.Sign}(\mathrm{SK}_{\mathsf{SIG}}, f)$ as is done honestly and returning $\mathrm{SK}(f) = (f, \sigma)$ to $\mathcal{A}$.
  - *Decryption key query*: Using $\mathrm{SK}^{(1)}_{\mathsf{PKE}}$, $\mathcal{C}$ answers a decryption key query of $\mathcal{A}$ corresponding to a decryption function $g \in \mathbb{G}$ by returning $\mathrm{DK}(g) = \left(g, \mathcal{O}(P^{(g, \mathrm{SK}^{(1)}_{\mathsf{PKE}}, \mathrm{MPK})})\right)$ to $\mathcal{A}$.
  - *Signcryption query*: When $\mathcal{A}$ queries a signcryption for $f \in \mathbb{F}$ along with $z \in \mathbb{D}_f$, $\mathcal{C}$ first generates $\sigma \leftarrow \mathsf{SIG.Sign}(\mathrm{SK}_{\mathsf{SIG}}, f)$ and $e_\ell = \mathsf{PKE.Encryt}(\mathrm{PK}^{(\ell)}_{\mathsf{PKE}}, f(z); r_\ell)$ using randomness r_ℓ for $\ell = 1, 2$. Then $\mathcal{C}$ generates a proof $\pi \leftarrow \mathsf{SSS\text{-}NIZKPoK.Prove}(\mathrm{CRS}, X, W)$ where $X = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1, e_2)$ and $W = (f(z), r_1, r_2, f, \sigma, z)$. $\mathcal{C}$ gives $\mathrm{CT} = (e_1, e_2, \pi)$ to $\mathcal{A}$.
  - *Unsigncryption query*: Since $\mathcal{C}$ can generate the decryption key corresponding to any decryption function $g \in \mathbb{G}$, it replies to an unsigncryption query of $\mathcal{A}$ for a ciphertext CT under some $g \in \mathbb{G}$, by preparing the decryption key $\mathrm{DK}(g)$ and returning the result of unsigncrypting CT using $\mathrm{DK}(g)$.
- $\mathcal{A}$ eventually outputs a forged ciphertext $\mathrm{CT}^* = (e_1^*, e_2^*, \pi^*)$. $\mathcal{C}$ verifies whether CT^* is indeed a valid forgery on m^*. In order to perform this verification, $\mathcal{C}$ has to check whether $\mathsf{SSS\text{-}NIZKPoK.Verify}(\mathrm{CRS}, (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1^*, e_2^*), \pi^*) = 1$, $\mathsf{PKE.Decrypt}(\mathrm{SK}^{(1)}_{\mathsf{PKE}}, e_1^*) = m^* = \mathsf{PKE.Decrypt}(\mathrm{SK}^{(2)}_{\mathsf{PKE}}, e_2^*)$, and m^* is actually not in the range of any function $f \in \mathbb{F}$ that $\mathcal{A}$ requested signing-keys for, as well as, $f(z) \neq m^*$ for all signcryption queries (f, z) made by $\mathcal{A}$. Observe that this verification process may not be efficient as verifying whether a

particular message is in the range of some function may not be possible in polynomial time. If the forgery verifies, then $\mathcal{C}$ outputs 1. Otherwise, it outputs 0.

Clearly, if $\mathcal{B}$ used $\mathsf{SSS\text{-}NIZKPoK.Setup}$ to generate CRS', then we are exactly in Hyb_0. On the other hand, if $\mathcal{B}$ used $\mathsf{SSS\text{-}NIZKPoK.ExtSetup}$ instead, then we are exactly in Hyb_1. Hence,

$$\begin{aligned}&\left|\mathsf{Forge}_0 - \mathsf{Forge}_1\right| \\ &= \left|\mathsf{Pr}\left[\text{CRS}' \leftarrow \mathsf{SSS\text{-}NIZKPoK.Setup}(1^\lambda) \ : \ \mathcal{C}(\text{CRS}') = 1\right]\right. \\ &\quad \left. - \mathsf{Pr}\left[(\text{CRS}', \widehat{\text{TR}}) \leftarrow \mathsf{SSS\text{-}NIZKPoK.ExtSetup}(1^\lambda) \ : \ \mathcal{C}(\text{CRS}') = 1\right]\right| = \mathsf{Adv}_{\mathcal{C}}^{\mathsf{SSS\text{-}NIZKPoK,CRS}}(\lambda).\end{aligned}$$

Thus we can see that if Forge_0 and Forge_1 are non-negligibly different, then $\mathcal{C}$ distinguishes between the common reference string generated by $\mathsf{SSS\text{-}NIZKPoK.Setup}$ and that formed by $\mathsf{SSS\text{-}NIZKPoK.ExtSetup}$ with non-negligible advantage. □

Lemma 8. *Assuming existential unforgeability against* CMA *of the signature scheme* SIG *and the extraction property of the* SSS-NIZKPoK *system, no* PPT *adversary has non-negligible advantage in* Hyb_1.

Proof. Consider a PPT adversary $\mathcal{A}$ for which Forge_1 is non-negligible. We construct an PPT algorithm $\mathcal{C}$ that breaks the existential unforgeability property of the signature scheme SIG using $\mathcal{A}$ as a subroutine. $\mathcal{C}$ interacts with $\mathcal{A}$ as follows:

- $\mathcal{C}$ begins by initializing $\mathcal{A}$ and receiving m^* from $\mathcal{A}$.
- In order to setup the public parameters, $\mathcal{C}$ performs the following steps:
  - $\mathcal{C}$ generates $(\text{PK}_{\mathsf{PKE}}^{(1)}, \text{SK}_{\mathsf{PKE}}^{(1)}), (\text{PK}_{\mathsf{PKE}}^{(2)}, \text{SK}_{\mathsf{PKE}}^{(2)}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$.
  - $\mathcal{C}$ obtains a verification key $\text{VK}'_{\mathsf{SIG}}$ for the signature scheme SIG from its *unforgeability challenger* $\mathcal{B}$ and designates it as VK_{SIG}.
  - Next $\mathcal{C}$ generates $(\text{CRS}, \widehat{\text{TR}}) \leftarrow \mathsf{SSS\text{-}NIZKPoK.ExtSetup}(1^\lambda)$.
  - $\mathcal{C}$ hands the public parameters $\text{MPK} = (\text{PK}_{\mathsf{PKE}}^{(1)}, \text{PK}_{\mathsf{PKE}}^{(2)}, \text{VK}_{\mathsf{SIG}}, \text{CRS})$ to $\mathcal{A}$ while keeps $\widehat{\text{MSK}} = (\text{SK}_{\mathsf{PKE}}^{(1)}, \widehat{\text{TR}})$.
- The signing key, decryption key, signcryption, and unsigncryption queries of $\mathcal{A}$ are answered by $\mathcal{C}$ as follows:
  - *Signing key query*: When $\mathcal{A}$ makes a signing key query corresponding to a signing function $f \in \mathbb{F}$, $\mathcal{C}$ forwards f to $\mathcal{B}$ and gets back a signature σ' on f. $\mathcal{C}$ returns $\text{SK}(f) = (f, \sigma = \sigma')$ to $\mathcal{A}$.
  - *Decryption key query*: Using $\text{SK}_{\mathsf{PKE}}^{(1)}$, $\mathcal{C}$ answers the decryption key queries of $\mathcal{A}$ in an identical manner as in the proof of Lemma 7.
  - *Signcryption query*: Note that the *constant functions* f_m (such that $f_m(z) = m$ for all $z \in \mathbb{D}_f$), for all $m \in \mathbb{M}$, are obviously contained in P/poly. Therefore, when $\mathcal{A}$ makes a signcryption query corresponding to signing function $f \in \mathbb{F}$ and $z \in \mathbb{D}_f$, $\mathcal{C}$ queries $\mathcal{B}$ with the function f_m, where $f(z) = m \in \mathbb{M}$, and receives back a signature σ'' on f_m. Then it computes $e_\ell = \mathsf{PKE.Encrypt}(\text{PK}_{\mathsf{PKE}}^{(\ell)}, m; r_\ell)$ using randomness r_ℓ, for

$\ell = 1, 2$. After that, it generates a proof $\pi \leftarrow \mathsf{SSS\text{-}NIZKPoK.Prove}(\mathrm{CRS}, X, W)$ where $X = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1, e_2)$ and $W = (m, r_1, r_2, f_m, \sigma'', z)$. $\mathcal{C}$ provides $\mathrm{CT} = (e_1, e_2, \pi)$ to $\mathcal{A}$. Note that CT is indeed a valid signcryption of $f(z) = m$.

– *Unsigncryption query*: The unsigncryption queries of $\mathcal{A}$ are answered by $\mathcal{C}$ in the same fashion as in the proof of Lemma 7.

- $\mathcal{A}$ eventually outputs a forgery $\mathrm{CT}^* = (e_1^*, e_2^*, \pi^*)$. Clearly, if CT^* is a valid signcryption of some message, then $\mathsf{SSS\text{-}NIZKPoK.Verify}(\mathrm{CRS}, X^*, \pi^*) = 1$ must hold, where $X^* = (\mathrm{PK}^{(1)}_{\mathsf{PKE}}, \mathrm{PK}^{(2)}_{\mathsf{PKE}}, \mathrm{VK}_{\mathsf{SIG}}, e_1^*, e_2^*)$. $\mathcal{C}$ performs $W^* \leftarrow \mathsf{SSS\text{-}NIZKPoK.Extr}(\mathrm{CRS}, \widehat{\mathrm{TR}}, X^*, \pi^*)$. Note that by the extraction property of $\mathsf{SSS\text{-}NIZKPoK}$, (X^*, W^*) must be a valid statement-witness pair of the relation R defined in equation (2) with all but negligible probability. Let $W^* = (m'^*, r_1^*, r_2^*, f^*, \sigma^*, z^*)$. $\mathcal{C}$ outputs (f^*, σ^*) as a forgery in its unforgeability game.

Observe that, if $\mathcal{A}$ indeed wins in the above experiment, i.e., CT^* is actually a valid signcryption of m^*, then we must have $m'^* = m^*$ and $f^*(z^*) = m'^* = m^*$. Now, by the rules of the game, for all $f \in \mathbb{F}$ on which $\mathcal{A}$ made a signing key query there does not exist any $z \in \mathbb{D}_f$ such that $f(z) = m^*$ and for any (f, z) pair on which a signcryption query was made by $\mathcal{A}$, it must hold that $f(z) \neq m^*$. Hence, it must be the case that $f^* \neq f$ for all $f \in \mathbb{F}$ on which a signing key query was made by $\mathcal{A}$. Further, recall that according to our simulation, for all the signcryption queries of $\mathcal{A}$ corresponding to a (f, z) pair, $\mathcal{C}$ used the *constant function* f_m, where $f(z) = m$, to provide a signcryption of m. But as m must be different from m^* by the rules of the game, $f^* \neq f_m$ either. Now, only the f's corresponding to $\mathcal{A}$'s signing key queries and f_m's with $m = f(z)$ corresponding to $\mathcal{A}$'s signcryption queries (f, z) are submitted by $\mathcal{C}$ to $\mathcal{B}$ for obtaining a signature. Therefore, (f^*, σ^*) is indeed a valid forgery produced by $\mathcal{C}$ in its unforgeability game. Thus, we have

$$\begin{aligned}
\mathsf{Forge}_1 \leq \mathsf{Pr}\big[(X^*, \pi^*) \leftarrow \mathcal{A}(\mathrm{CRS});\ W^* \leftarrow \mathsf{SSS\text{-}NIZKPoK.Extr}(\mathrm{CRS}, \widehat{\mathrm{TR}}, X^*, \pi^*)\ :\\
\mathsf{SSS\text{-}NIZKPoK.Verify}(\mathrm{CRS}, X^*, \pi^*) = 1 \bigwedge (X^*, W^*) \notin R\big] +\\
\mathsf{Pr}\big[(\mathrm{VK}'_{\mathsf{SIG}}, \mathrm{SK}'_{\mathsf{SIG}}) \leftarrow \mathsf{SIG.KeyGen}(1^\lambda); (f^*, \sigma^*) \leftarrow \mathcal{C}^{\mathsf{O_S}(\cdot)}(\mathrm{VK}'_{\mathsf{SIG}})\ :\\
\mathsf{SIG.Verify}(\mathrm{VK}'_{\mathsf{SIG}}, f^*, \sigma^*) = 1 \bigwedge f^* \notin Q\big]\\
= \mathsf{Adv}^{\mathsf{SSS\text{-}NIZKPoK,EXT}}_{\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{SIG,UF\text{-}CMA}}_{\mathcal{C}}(\lambda),
\end{aligned}$$

where $\mathsf{O_S}(f) = \mathsf{SIG.Sign}(\mathrm{SK}'_{\mathsf{SIG}}, f)$, Q represents the list of queries of $\mathcal{C}$ to $\mathsf{O_S}$, and $\mathsf{Adv}^{\mathsf{SIG,UF\text{-}CMA}}_{\mathcal{C}}(\lambda)$ denotes the advantage of $\mathcal{C}$ in forging a signature for SIG. The lemma follows. □

□

5 Attribute-Based Signcryption for General Circuits from FSC

5.1 The Notion of Attribute-Based Signcryption for General Circuits

The formal notion of *attribute-based signcryption* (ABSC) [GNSN10], [RD14b], [RD14a], [WH11], when supporting *general polynomial-size circuits*, can be described as follows: As for other attribute-based primitives such as attribute-based encryption or signature, ABSC also has two flavors, namely, *key-policy* and *ciphertext-policy*. For definiteness of exposure, lets consider the key-policy version where the access structures, represented as circuits, are embedded in signing and decryption keys, while signing and decryption attribute sets, expressed as bit strings, are associated with signcrypted message.

Let $\mathbb{F}_{\mathsf{ABSC}} \subset \mathsf{P/poly}$ and $\mathbb{G}_{\mathsf{ABSC}} \subset \mathsf{P/poly}$ respectively denote the class of Boolean circuits representing the signing and decryption policies with the input strings to these circuits representing the signing and decryption attribute sets. Let $\mathbb{M}_{\mathsf{ABSC}} \subset \{0,1\}^*$ be the message space

and λ be the underlying security parameter. A key-policy ABSC scheme consists of the following PPT algorithms:

$\mathsf{ABSC.Setup}(1^\lambda)$: The trusted authority publishes the public parameters $\mathrm{MPK}_{\mathsf{ABSC}}$, while keeps the master secret key $\mathrm{MSK}_{\mathsf{ABSC}}$ to itself.

$\mathsf{ABSC.SKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{SIG})})$: The authority hands a signing key $\mathrm{SK}_{\mathsf{ABSC}}(C^{(\mathsf{SIG})})$ for a signing circuit $C^{(\mathsf{SIG})} \in \mathbb{F}_{\mathsf{ABSC}}$ with input length $\mu(\lambda)$, for some polynomial μ, to a signcrypter.

$\mathsf{ABSC.DKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{DEC})})$: A decryption key $\mathrm{DK}_{\mathsf{ABSC}}(C^{(\mathsf{DEC})})$ corresponding to a decryption circuit $C^{(\mathsf{DEC})} \in \mathbb{G}_{\mathsf{ABSC}}$ having input length $\nu(\lambda)$ is given to a decrypter by the trusted authority, where ν is some polynomial.

$\mathsf{ABSC.Signcrypt}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{SK}_{\mathsf{ABSC}}(C^{(\mathsf{SIG})}), y, \overline{y}, M)$: A signcrypter outputs a signcryption $\mathrm{CT}_{\mathsf{ABSC}}^{(y,\overline{y})}$ of a message $M \in \mathbb{M}_{\mathsf{ABSC}}$ under decryption input string $y \in \{0,1\}^\nu$ and signature input string $\overline{y} \in \{0,1\}^\mu$.

$\mathsf{ABSC.Unsigncrypt}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{DK}_{\mathsf{ABSC}}(C^{(\mathsf{DEC})}), \mathrm{CT}_{\mathsf{ABSC}}^{(y,\overline{y})})$: A decrypter ends up either retrieving the signcrypted message or $\perp$ if unsuccessful.

A key-policy ABSC scheme is *correct* if for all $C^{(\mathsf{SIG})} \in \mathbb{F}_{\mathsf{ABSC}}$ with input length μ representing signing access structures, $\overline{y} \in \{0,1\}^\mu$ representing signing attribute sets with $C^{(\mathsf{SIG})}(\overline{y}) = 1$, $C^{(\mathsf{DEC})} \in \mathbb{G}_{\mathsf{ABSC}}$ with input length ν expressing decryption access policies, $y \in \{0,1\}^\nu$ expressing decryption attribute sets such that $C^{(\mathsf{DEC})}(y) = 1$, and $M \in \mathbb{M}_{\mathsf{ABSC}}$,

$$\begin{aligned}&\mathsf{Pr}\big[(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}) \leftarrow \mathsf{ABSC.Setup}(1^\lambda) \ : \\ &\quad \mathsf{ABSC.Unsigncrypt}\big(\mathrm{MPK}_{\mathsf{ABSC}}, \mathsf{ABSC.DKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{DEC})}), \\ &\quad \mathsf{ABSC.Signcrypt}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathsf{ABSC.SKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{SIG})}), y, \overline{y}, M)\big) = M\big] \\ &\quad > 1 - \epsilon(\lambda)\end{aligned}$$

for some negligible function ϵ. Here $C^{(\mathsf{SIG})}(\overline{y}) = 1$ indicates that the signing access structure represented by $C^{(\mathsf{SIG})}$ is satisfied by the signing attribute set expressed by $\overline{y}$. $C^{(\mathsf{DEC})}(y) = 1$ has the similar meaning.

A key-policy ABSC scheme has two security requirements: (I) *message confidentiality* against CPA and (II) *ciphertext unforgeability* against CMA. The selective versions of these security notions are defined as follows:

(I) ***Message Confidentiality***: An ABSC scheme is said to be selectively message confidential against CPA if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function ϵ such that for any security parameter λ,

$$\begin{aligned}&\big|\mathsf{Pr}\big[(y^*, \overline{y}^*, M_0^*, M_1^*) \leftarrow \mathcal{A}_1(1^\lambda);\ (\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}) \leftarrow \mathsf{ABSC.Setup}(1^\lambda);\ b \leftarrow \{0,1\}; \\ &\quad \mathrm{CT}_{\mathsf{ABSC}}^* \leftarrow \mathsf{ABSC.Signcrypt}\big(\mathrm{MPK}_{\mathsf{ABSC}}, \mathsf{ABSC.SKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{SIG})^*}), y^*, \overline{y}^*, M_b^*\big); \\ &\quad b' \leftarrow \mathcal{A}_2^{\mathsf{O}_{\mathsf{SK}}(\cdot), \mathsf{O}_{\mathsf{DK}}(\cdot), \mathsf{O}_{\mathsf{SC}}(\cdot,\cdot,\cdot)}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{CT}_{\mathsf{ABSC}}^*) \ : \ C^{(\mathsf{SIG})^*}(\overline{y}^*) = 1 \ \bigwedge \\ &\quad C^{(\mathsf{DEC})}(y^*) = 0 \ \forall \ C^{(\mathsf{DEC})} \in Q \ \bigwedge \ b' = b\big] - 1/2\big| < \epsilon(\lambda),\end{aligned}$$

where

$$\mathsf{O}_{\mathsf{SK}}(C^{(\mathsf{SIG})}) = \mathsf{ABSC.SKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{SIG})}),$$
$$\mathsf{O}_{\mathsf{DK}}(C^{(\mathsf{DEC})}) = \mathsf{ABSC.DKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{DEC})}),$$
$$\mathsf{O}_{\mathsf{SC}}(y, \overline{y}, M) = \mathsf{ABSC.Signcrypt}\big(\mathrm{MPK}_{\mathsf{ABSC}}, \mathsf{ABSC.SKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{SIG})}), y, \overline{y}, M\big) \text{ with } C^{(\mathsf{SIG})}(\overline{y}) = 1,$$

and Q represents the list of queries made by $\mathcal{A}_2$ to O_{DK}. Note that here we only consider post-challenge queries. However, as noted in [GGH+13b], this does not weaken the selective security notion.

(II) ***Ciphertext Unforgeability***: An ABSC scheme is said to be selectively ciphertext unforgeable against CMA if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function ϵ such that for any security parameter λ,

$$\mathsf{Pr}\big[(y^*, \overline{y}^*, M^*) \leftarrow \mathcal{A}_1(1^\lambda);\ (\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}) \leftarrow \mathsf{ABSC.Setup}(1^\lambda);$$
$$\mathrm{CT}^*_{\mathsf{ABSC}} \leftarrow \mathcal{A}_2^{\mathsf{O}_{\mathsf{SK}}(\cdot),\mathsf{O}_{\mathsf{DK}}(\cdot),\mathsf{O}_{\mathsf{SC}}(\cdot,\cdot,\cdot),\mathsf{O}_{\mathsf{US}}(\cdot,\cdot)}(\mathrm{MPK}_{\mathsf{ABSC}}) :$$
$$M^* \leftarrow \mathsf{ABSC.Unsigncrypt}\big(\mathrm{MPK}_{\mathsf{ABSC}}, \mathsf{ABSC.DKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{DEC})}), \mathrm{CT}^*_{\mathsf{ABSC}}\big)$$
$$\forall\ C^{(\mathsf{DEC})} \text{ with } C^{(\mathsf{DEC})}(y^*) = 1 \bigwedge M^* \neq \perp \bigwedge C^{(\mathsf{SIG})}(\overline{y}^*) = 0\ \forall\ C^{(\mathsf{SIG})} \in Q_1 \bigwedge$$
$$(\overline{y}, M) \neq (\overline{y}^*, M^*)\ \forall\ (\overline{y}, M) \in Q_2\big] < \epsilon(\lambda),$$

where

$$\mathsf{O}_{\mathsf{SK}}(C^{(\mathsf{SIG})}) = \mathsf{ABSC.SKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{SIG})}),$$
$$\mathsf{O}_{\mathsf{DK}}(C^{(\mathsf{DEC})}) = \mathsf{ABSC.DKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{DEC})}),$$
$$\mathsf{O}_{\mathsf{SC}}(y, \overline{y}, M) = \mathsf{ABSC.Signcrypt}\big(\mathrm{MPK}_{\mathsf{ABSC}}, \mathsf{ABSC.SKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{SIG})}), y, \overline{y}, M\big) \text{ with } C^{(\mathsf{SIG})}(\overline{y}) = 1,$$
$$\mathsf{O}_{\mathsf{US}}(C^{(\mathsf{DEC})}, \mathrm{CT}^{(y,\overline{y})}_{\mathsf{ABSC}}) = \mathsf{ABSC.Unsigncrypt}\big(\mathrm{MPK}_{\mathsf{ABSC}}, \mathsf{ABSC.DKeyGen}(\mathrm{MPK}_{\mathsf{ABSC}}, \mathrm{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{DEC})}), \mathrm{CT}^{(y,\overline{y})}_{\mathsf{ABSC}}\big) \text{ with } C^{(\mathsf{DEC})}(y) = 1,$$

Q_1 is the set of queries of $\mathcal{A}_2$ to O_{SK}, and Q_2 is the collection of pairs of signature input string and messages that are submitted along with some decryption input string to O_{SC} by $\mathcal{A}_2$.

5.2 Our Key-Policy ABSC Scheme

Let λ be the underlying security parameter. We consider the family of signing circuits $\mathbb{F}_{\mathsf{ABSC}} \subset \mathsf{P/poly}$ whose members have input length $\mu(\lambda)$ and the class of decryption circuits $\mathbb{G}_{\mathsf{ABSC}} \subset \mathsf{P/poly}$ containing circuits of input length $\nu(\lambda)$ for polynomials μ and ν. The message space for our ABSC scheme is $\mathbb{M}_{\mathsf{ABSC}} = \{0,1\}^{\gamma(\lambda)}$ for some polynomial γ. Consider a FSC scheme FSC=(FSC.Setup, FSC.SKeyGen, FSC.Signcrypt, FSC.DKeyGen,FSC.Unsigncrypt) supporting any polynomial-size signing function family $\mathbb{F} \subseteq \mathsf{P/poly}$, decryption function class $\mathbb{G} \subseteq \mathsf{P/poly}$, and the message space $\mathbb{M} = \{0,1\}^n \cup \{\perp\}$ where $n = \nu + \mu + \gamma$. Let us now define signing and decryption functions for the FSC scheme that would instantiate ABSC. We associate a signing function $f_{C^{(\mathsf{SIG})}} \in \mathbb{F}$, where $f_{C^{(\mathsf{SIG})}} : \mathbb{D}_f = \{0,1\}^n \rightarrow \mathbb{M}$ to each signing circuit $C^{(\mathsf{SIG})} \in \mathbb{F}_{\mathsf{ABSC}}$ and a decryption function $g_{C^{(\mathsf{DEC})}} \in \mathbb{G}$, where $g_{C^{(\mathsf{DEC})}} : \mathbb{M} \rightarrow \mathbb{M}$, corresponding to each decryption

circuit $C^{(\mathsf{DEC})} \in \mathbb{G}_{\mathsf{ABSC}}$, defined as follows:

$$f_{C^{(\mathsf{SIG})}}(y\|\overline{y}\|M) = \begin{cases} y\|\overline{y}\|M, \text{ if } C^{(\mathsf{SIG})}(\overline{y}) = 1 \\ \perp, \qquad \text{otherwise} \end{cases}$$
$$g_{C^{(\mathsf{DEC})}}(y\|\overline{y}\|M) = \begin{cases} y\|\overline{y}\|M, \text{ if } C^{(\mathsf{DEC})}(y) = 1 \\ \perp, \qquad \text{otherwise} \end{cases} \tag{3}$$

Our ABSC works as follows:

$\mathsf{ABSC.Setup}(1^\lambda)$: The trusted authority takes as input the security parameter λ and generates $(\text{MPK}, \text{MSK}) \leftarrow \mathsf{FSC.Setup}(1^\lambda)$. It publishes the public parameters $\text{MPK}_{\mathsf{ABSC}} = \text{MPK}$, while keeps the master secret key $\text{MSK}_{\mathsf{ABSC}} = \text{MSK}$ to itself.

$\mathsf{ABSC.SKeyGen}(\text{MPK}_{\mathsf{ABSC}} = \text{MPK}, \text{MSK}_{\mathsf{ABSC}} = \text{MSK}, C^{(\mathsf{SIG})})$: The authority creates the signing key $\text{SK}(f_{C^{(\mathsf{SIG})}}) \leftarrow \mathsf{FSC.SKeyGen}(\text{MPK}, \text{MSK}, f_{C^{(\mathsf{SIG})}})$ for the signing function $f_{C^{(\mathsf{SIG})}} \in \mathbb{F}$ defined in equation (3) and provides $\text{SK}_{\mathsf{ABSC}}(C^{(\mathsf{SIG})}) = \text{SK}(f_{C^{(\mathsf{SIG})}})$ to the signcrypter.

$\mathsf{FSC.DKeyGen}(\text{MPK}_{\mathsf{ABSC}} = \text{MPK}, \text{MSK}_{\mathsf{ABSC}} = \text{MSK}, C^{(\mathsf{DEC})})$: The trusted authority prepares the decryption key $\text{DK}(g_{C^{(\mathsf{DEC})}}) \leftarrow \mathsf{FSC.DKeyGen}(\text{MPK}, \text{MSK}, g_{C^{(\mathsf{DEC})}})$ for the decryption function $g_{C^{(\mathsf{DEC})}} \in \mathbb{G}$ described in equation (3) and gives $\text{DK}_{\mathsf{ABSC}}(C^{(\mathsf{DEC})}) = \text{DK}(g_{C^{(\mathsf{DEC})}})$ to the decrypter.

$\mathsf{ABSC.Signcrypt}(\text{MPK}_{\mathsf{ABSC}} = \text{MPK}, \text{SK}_{\mathsf{ABSC}}(C^{(\mathsf{SIG})}) = \text{SK}(f_{C^{(\mathsf{SIG})}}), y, \overline{y}, M)$: Provided $C^{(\mathsf{SIG})}(\overline{y}) = 1$, a signcrypter computes $\text{CT} \leftarrow \mathsf{FSC.Signcrypt}(\text{MPK}, \text{SK}(f_{C^{(\mathsf{SIG})}}), z = y\|\overline{y}\|M)$. It outputs $\text{CT}^{(y,\overline{y})}_{\mathsf{ABSC}} = (y, \overline{y}, \text{CT})$.

$\mathsf{ABSC.Unsigncrypt}(\text{MPK}_{\mathsf{ABSC}} = \text{MPK}, \text{DK}_{\mathsf{ABSC}}(C^{(\mathsf{DEC})}) = \text{DK}(g_{C^{(\mathsf{DEC})}}), \text{CT}^{(y,\overline{y})}_{\mathsf{ABSC}} = (y, \overline{y}, \text{CT}))$: A decrypter runs $\mathsf{FSC.Unsigncrypt}(\text{MPK}, \text{DK}(g_{C^{(\mathsf{DEC})}}), \text{CT})$ and obtains $y'\|\overline{y}'\|M'$ or $\perp$. If the decrypter gets $y'\|\overline{y}'\|M'$ and it holds that $y' = y \ \bigwedge \ \overline{y}' = \overline{y}$, then the decrypter outputs M'. Otherwise, it outputs $\perp$.

Note that the correctness of the above ABSC scheme is immediate from the correctness of the FSC scheme. The security follows from the following two theorems:

Theorem 3 (Message Confidentiality of ABSC). *If* FSC *is selectively message confidential against* CPA *as per the definition of §3, then the* ABSC *described above is also selectively message confidential against* CPA *as per the notion given in §5.1.*

Proof. Assume that there is a PPT adversary $\mathcal{A}$ that breaks the selective CPA-message confidentiality of the above ABSC scheme. We construct a PPT adversary $\mathcal{C}$ that breaks the selective CPA-message confidentiality of FSC using $\mathcal{A}$ as a subroutine. $\mathcal{C}$ interacts with $\mathcal{A}$ as follows:

- $\mathcal{C}$ begins by initializing $\mathcal{A}$ and receiving $(y^*, \overline{y}^*, M_0^*, M_1^*)$ from $\mathcal{A}$. $\mathcal{C}$ determines a signing circuit $C^{(\mathsf{SIG})^*} \in \mathbb{F}_{\mathsf{ABSC}}$ such that $C^{(\mathsf{SIG})^*}(\overline{y}^*) = 1$ and constructs the signing function $f^* = f_{C^{(\mathsf{SIG})^*}} \in \mathbb{F}$. $\mathcal{C}$ submits the two challenge pairs $(f_0^*, z_0^*) = (f^*, y^*\|\overline{y}^*\|M_0^*), (f_1^*, z_1^*) = (f^*, y^*\|\overline{y}^*\|M_1^*)$ to its *message confidentiality challenger* $\mathcal{B}$ for FSC.

- $\mathcal{C}$ gets the public parameters MPK from $\mathcal{B}$ and provides $\text{MPK}_{\mathsf{ABSC}} = \text{MPK}$ to $\mathcal{A}$.

- Upon receiving a challenge ciphertext CT^* from $\mathcal{B}$, $\mathcal{C}$ hands to $\mathcal{A}$ the challenge ciphertext $\text{CT}^*_{\mathsf{ABSC}} = (y^*, \overline{y}^*, \text{CT}^*)$.

- The signing key, decryption key, and signcryption queries of $\mathcal{A}$ are answered by $\mathcal{C}$ as follows:

– *Signing key query*: When $\mathcal{A}$ makes a signing key query for a signing circuit $C^{(\mathsf{SIG})} \in \mathbb{F}_{\mathsf{ABSC}}$, $\mathcal{C}$ queries $\mathcal{B}$ with the function $f_{C^{(\mathsf{SIG})}} \in \mathbb{F}$ constructed from $C^{(\mathsf{SIG})}$ and obtains $\mathrm{SK}(f_{C^{(\mathsf{SIG})}})$. $\mathcal{C}$ hands $\mathrm{SK}_{\mathsf{ABSC}}(C^{(\mathsf{SIG})}) = \mathrm{SK}(f_{C^{(\mathsf{SIG})}})$ to $\mathcal{A}$.

– *Decryption key query*: When $\mathcal{A}$ queries $\mathcal{C}$ with a decryption circuit $C^{(\mathsf{DEC})} \in \mathbb{G}_{\mathsf{ABSC}}$, subject to the constraint $C^{(\mathsf{DEC})}(y^*) = 0$, $\mathcal{C}$ queries $\mathcal{B}$ with the function $g_{C^{(\mathsf{DEC})}} \in \mathbb{G}$ constructed from $C^{(\mathsf{DEC})}$. Note that since $C^{(\mathsf{DEC})}(y^*) = 0$, we have $g_{C^{(\mathsf{DEC})}}(f_0^*(z_0^*)) = \perp = g_{C^{(\mathsf{DEC})}}(f_1^*(z_1^*))$, thereby, the restriction on the decryption key query in the selective CPA-message confidentiality game for FSC is satisfied. $\mathcal{B}$ returns a decryption key $\mathrm{DK}(g_{C^{(\mathsf{DEC})}})$ to $\mathcal{C}$ and $\mathcal{C}$ gives $\mathrm{DK}_{\mathsf{ABSC}}(C^{(\mathsf{DEC})}) = \mathrm{DK}(g_{C^{(\mathsf{DEC})}})$ to $\mathcal{A}$.

– *Signcryption query*: Upon receiving a signcryption query from $\mathcal{A}$ for a tuple $(y, \overline{y}, M)$, $\mathcal{C}$ identifies a circuit $C^{(\mathsf{SIG})} \in \mathbb{F}_{\mathsf{ABSC}}$ with $C^{(\mathsf{SIG})}(\overline{y}) = 1$ and queries $\mathcal{B}$ with the pair (f, z), where $f = f_{C^{(\mathsf{SIG})}} \in \mathbb{F}$ is constructed from $C^{(\mathsf{SIG})}$ and $z = y\|\overline{y}\|M$. $\mathcal{B}$ returns the ciphertext CT to $\mathcal{C}$ and $\mathcal{C}$ gives $\mathrm{CT}_{\mathsf{ABSC}}^{(y,\overline{y})} = (y, \overline{y}, \mathrm{CT})$ to $\mathcal{A}$.

- Eventually $\mathcal{A}$ outputs a bit b'. $\mathcal{C}$ also outputs b'.

Clearly the simulation is perfect. Thus, the advantage of $\mathcal{A}$ in the selective CPA-message confidentiality game for ABSC is equal to that of $\mathcal{C}$ in the selective CPA-message confidentiality game for FSC. Hence the theorem. □

Theorem 4 (Ciphertext Unforgeability of ABSC). *If the* FSC *scheme is selectively ciphertext unforgeable against* CMA *as per the definition of §3, then the* ABSC *scheme described above is also selectively ciphertext unforgeable against* CMA *according to the notion given in §5.1.*

Proof. Suppose there is a PPT adversary $\mathcal{A}$ that breaks the selective CMA-ciphertext unforgeability of the ABSC scheme described above. We construct a PPT adversary $\mathcal{C}$ that breaks the selective CMA-ciphertext unforgeability of the FSC scheme using $\mathcal{A}$ as a subroutine. $\mathcal{C}$ interacts with $\mathcal{A}$ as follows:

- $\mathcal{C}$ starts by initializing $\mathcal{A}$ and receiving $(y^*, \overline{y}^*, M^*)$ from $\mathcal{A}$. $\mathcal{C}$ submits $m^* = y^*\|\overline{y}^*\|M^*$ to its *unforgeability challenger* $\mathcal{B}$ for the FSC scheme.

- The different types of queries of $\mathcal{A}$ are answered by $\mathcal{C}$ as described below:

  – *Signing key query*: When $\mathcal{A}$ queries a signing key for a circuit $C^{(\mathsf{SIG})} \in \mathbb{F}_{\mathsf{ABSC}}$ subject to the restriction that $C^{(\mathsf{SIG})}(\overline{y}^*) = 0$, $\mathcal{C}$ queries $\mathcal{B}$ with the function $f_{C^{(\mathsf{SIG})}} \in \mathbb{F}$ constructed from $C^{(\mathsf{SIG})}$. Note that since $C^{(\mathsf{SIG})}(\overline{y}^*) = 0$, $f_{C^{(\mathsf{SIG})}}(y\|\overline{y}^*\|M) = \perp$ for any y and M, and hence, in particular $y^*\|\overline{y}^*\|M^*$ is not in the range of $f_{C^{(\mathsf{SIG})}}$. Thus, the restriction on the signing key queries in the selective CMA-ciphertext unforgeability game for FSC is satisfied. $\mathcal{B}$ returns $\mathrm{SK}(f_{C^{(\mathsf{SIG})}})$ to $\mathcal{C}$ and $\mathcal{C}$ passes it to $\mathcal{A}$ as $\mathrm{SK}_{\mathsf{ABSC}}(C^{(\mathsf{SIG})})$.

  – *Decryption key query*: Upon receiving a decryption key query from $\mathcal{A}$ corresponding to a decryption circuit $C^{(\mathsf{DEC})} \in \mathbb{G}_{\mathsf{ABSC}}$, $\mathcal{C}$ constructs the function $g_{C^{(\mathsf{DEC})}} \in \mathbb{G}$ from $C^{(\mathsf{DEC})}$ and queries $\mathcal{B}$ with $g_{C^{(\mathsf{DEC})}}$. The decryption key $\mathrm{DK}(g_{C^{(\mathsf{DEC})}})$ returned by $\mathcal{B}$ is handed over to $\mathcal{A}$ as $\mathrm{DK}_{\mathsf{ABSC}}(C^{(\mathsf{DEC})})$ by $\mathcal{C}$.

  – *Signcryption query*: Upon receiving a signcryption query from $\mathcal{A}$ for a tuple $(y, \overline{y}, M)$ such that $(\overline{y}, M) \neq (\overline{y}^*, M^*)$, $\mathcal{C}$ identifies a signing circuit $C^{(\mathsf{SIG})} \in \mathbb{F}_{\mathsf{ABSC}}$ with $C^{(\mathsf{SIG})}(\overline{y}) = 1$. Then $\mathcal{C}$ constructs the signing function $f_{C^{(\mathsf{SIG})}} \in \mathbb{F}$ from $C^{(\mathsf{SIG})}$ and queries $\mathcal{B}$ with $(f_{C^{(\mathsf{SIG})}}, y\|\overline{y}\|M)$. Note that since $(\overline{y}, M) \neq (\overline{y}^*, M^*)$, we have $y\|\overline{y}\|M \neq y^*\|\overline{y}^*\|M^*$ and so $f_{C^{(\mathsf{SIG})}}(y\|\overline{y}\|M) \neq y^*\|\overline{y}^*\|M^*$. Thus, the restriction on the signcryption query in selective

CMA-ciphertext unforgeability game for FSC is satisfied. $\mathcal{B}$ returns back a ciphertext CT to $\mathcal{C}$ and $\mathcal{C}$ gives $\mathcal{A}$ the ciphertext $\text{CT}^{(y,\overline{y})}_{\mathsf{ABSC}} = (y, \overline{y}, \text{CT})$.

– *Unsigncryption query*: In response to an unsigncryption query of $\mathcal{A}$ for a decryption circuit $C^{(\mathsf{DEC})} \in \mathbb{G}_{\mathsf{ABSC}}$ and ciphertext $\text{CT}^{(y,\overline{y})}_{\mathsf{ABSC}} = (y, \overline{y}, \text{CT})$, $\mathcal{C}$ constructs the decryption function $g_{C^{(\text{DEC})}} \in \mathbb{G}$ from $C^{(\mathsf{DEC})}$ and queries $\mathcal{B}$ with $(\text{CT}, g_{C^{(\text{DEC})}})$. If $\mathcal{B}$ returns $y' \| \overline{y}' \| M$ and it holds that $y' = y \ \bigwedge \ \overline{y}' = \overline{y}$, then $\mathcal{C}$ gives M to $\mathcal{A}$. On the other hand, if $y' \neq y \ \bigvee \ \overline{y}' \neq \overline{y}$ or $\mathcal{B}$ returns $\perp$, then $\mathcal{C}$ provides $\perp$ to $\mathcal{A}$.

- Eventually, $\mathcal{A}$ outputs a forgery $\text{CT}^*_{\mathsf{ABSC}} = (y^*, \overline{y}^*, \text{CT}^*)$ on M^* under y^* and $\overline{y}^*$. $\mathcal{C}$ outputs CT^* as a forgery in its unforgeability game.

Observe that if $\text{CT}^*_{\mathsf{ABSC}}$ is a valid forgery on $(y^*, \overline{y}^*, M^*)$, then it must hold that $M^*(\neq \perp) \leftarrow \mathsf{ABSC.Unsigncrypt}(\text{MPK}_{\mathsf{ABSC}}, \mathsf{ABSC.DKeyGen}(\text{MPK}_{\mathsf{ABSC}}, \text{MSK}_{\mathsf{ABSC}}, C^{(\mathsf{DEC})}), \text{CT}^*_{\mathsf{ABSC}})$ for all $C^{(\mathsf{DEC})} \in \mathbb{G}_{\mathsf{ABSC}}$ with $C^{(\mathsf{DEC})}(y^*) = 1$, i.e., in other words, $y^* \| \overline{y}^* \| M^* \leftarrow \mathsf{FSC.Unsigncrypt}(\text{MPK}, \mathsf{FSC.DKeyGen}(\text{MPK}, \text{MSK}, g_{C^{(\text{DEC})}}), \text{CT}^*)$ for all $g_{C^{(\text{DEC})}} \in \mathbb{G}$ such that $C^{(\mathsf{DEC})}(y^*) = 1$. Since any $g_{C^{(\text{DEC})}}$ with $C^{(\mathsf{DEC})}(y^*) = 1$, outputs $y^* \| \overline{y}^* \| M^*$ if and only if the input is $y^* \| \overline{y}^* \| M^*$, CT^* must be a valid functional signcryption of $y^* \| \overline{y}^* \| M^*$. Thus, if $\mathcal{A}$ wins the selective CMA-ciphertext unforgeability game for ABSC, then $\mathcal{C}$ wins in the selective CMA-ciphertext unforgeability game for FSC. Hence the theorem. □

Remark 2 (Ciphertext-Policy ABSC from FSC). Note that we can also instantiate ciphertext-policy ABSC using FSC. In ciphertext-policy ABSC, signing and decryption keys are associated with strings and ciphertexts are attached to pairs of circuits expressing signature and decryption policies. Consider the class of signature policy circuits $\mathbb{F}_{\mathsf{ABSC}} \subset \mathsf{P/poly}$, the canonical string representation of members of which have length at most $\mu(\lambda)$, the class of decryption policy circuits $\mathbb{G}_{\mathsf{ABSC}} \subset \mathsf{P/poly}$ with members having canonical string representation of length $\nu(\lambda)$, and the message space $\mathbb{M}_{\mathsf{ABSC}} = \{0,1\}^{\gamma(\lambda)}$, for polynomials μ, ν, and γ, where λ is the underlying security parameter. Consider a FSC scheme FSC with message space $\mathbb{M} = \{0,1\}^n \cup \{\perp\}$, where $n = \mu + \nu + \gamma$, and class of signing, decryption functions $\mathbb{F}, \mathbb{G} \subseteq \mathsf{P/poly}$ respectively. We define signing function $f_{\overline{y}} \in \mathbb{F}$, where $f_{\overline{y}} : \{0,1\}^n \to \mathbb{M}$, for each signature input string $\overline{y}$ and decryption function $g_y \in \mathbb{G}$, where $g_y : \mathbb{M} \to \mathbb{M}$, corresponding to each decryption input string y that would instantiate the ciphertext-policy ABSC as follows:

$$
\begin{aligned}
f_{\overline{y}}(C^{(\mathsf{DEC})} \| C^{(\mathsf{SIG})} \| M) &= \begin{cases} C^{(\mathsf{DEC})} \| C^{(\mathsf{SIG})} \| M, & \text{if } C^{(\mathsf{SIG})}(\overline{y}) = 1 \\ \perp, & \text{otherwise} \end{cases} \\
g_y(C^{(\mathsf{DEC})} \| C^{(\mathsf{SIG})} \| M) &= \begin{cases} C^{(\mathsf{DEC})} \| C^{(\mathsf{SIG})} \| M, & \text{if } C^{(\mathsf{DEC})}(y) = 1 \\ \perp, & \text{otherwise} \end{cases}
\end{aligned} \tag{4}
$$

where $C^{(\mathsf{SIG})} \in \mathbb{F}_{\mathsf{ABSC}}$, $C^{(\mathsf{DEC})} \in \mathbb{G}_{\mathsf{ABSC}}$, and $M \in \mathbb{M}_{\mathsf{ABSC}}$. When consider in the arguments of $f_{\overline{y}}$ and g_y, $C^{(\mathsf{DEC})}$ and $C^{(\mathsf{SIG})}$ are taken in their canonical string representation form. With this setting, we can construct ciphertext-policy ABSC from FSC in an analogous fashion as has been done for the key-policy variant earlier in this section.

6 Other Cryptographic Primitives from FSC

§4.1 showed which cryptographic primitives are *sufficient* for FSC. We now seek the answer for the converse question: Which primitives are *necessary* for FSC? We show in this section that FSC implies statistically simulation-sound non-interactive zero-knowledge proof of knowledge (SSS-NIZKPoK) for any NP relation and indistinguishability obfuscation (IO) for P/poly.

6.1 SSS-NIZKPoK from FSC

Let R be an NP relation and $\mathbb{L}$ be the associated language. Recall that for $(X, W) \in R$, we call X the statement that is contained in $\mathbb{L}$ and W an witness for X. Let κ and ρ be the upper bounds on the lengths of the statements and witnesses of R. Consider a FSC scheme FSC=(FSC.Setup, FSC.SKeyGen, FSC.Signcrypt, FSC.DKeyGen, FSC.Unsigncrypt) supporting signing function family $\mathbb{F} \subseteq \mathsf{P/poly}$ and class of decryption functions $\mathbb{G} \subseteq \mathsf{P/poly}$. Let $\mathbb{M} = \{0,1\}^n \cup \{\perp\}$ be the message space of FSC where $n = \kappa + \rho + 1$. The SSS-NIZKPoK system is described as follows:

SSS-NIZKPoK.Setup(1^λ): The trusted authority runs $(\text{MPK}, \text{MSK}) \leftarrow$ FSC.Setup(1^λ). Next, it identifies some statement $X^* \in \mathbb{L}$. Then, it generates a signing key for FSC, $\text{SK}(f) \leftarrow$ FSC.SKeyGen$(\text{MPK}, \text{MSK}, f)$ for the signing function $f \in \mathbb{F}$, where $f : \{0,1\}^n \rightarrow \mathbb{M}$, and a decryption key for FSC, $\text{DK}(g) \leftarrow$ FSC.DKeyGen$(\text{MPK}, \text{MSK}, g)$ for the decryption function $g \in \mathbb{G}$, where $g : \mathbb{M} \rightarrow \{0,1\}^\kappa \cup \{\perp\}$, defined as follows:

$$f(X\|W\|\beta) = \begin{cases} X\|W\|\beta, \text{ if } (X,W) \in R \ \bigwedge \ \beta = 1 \\ \perp, \qquad \text{otherwise} \end{cases}$$
$$g(X\|W\|\beta) = \begin{cases} X, \text{ if } [(X,W) \in R \ \bigwedge \ \beta = 1] \ \bigvee \ [X = X^* \ \bigwedge \ W = 0^\rho \ \bigwedge \ \beta = 0] \\ \perp, \text{ otherwise} \end{cases} \tag{5}$$

The trusted authority publishes the common reference string $\text{CRS} = (\text{MPK}, \text{SK}(f), \text{DK}(g))$.

SSS-NIZKPoK.Prove(CRS, X, W): A prover executes $\text{CT} \leftarrow$ FSC.Signcrypt$(\text{MPK}, \text{SK}(f), X\|W\|1)$ and outputs $\pi = \text{CT}$.

SSS-NIZKPoK.Verify$(\text{CRS}, X, \pi = \text{CT})$: A verifier runs $X' \leftarrow$ FSC.Unsigncrypt$(\text{MPK}, \text{DK}(g), \text{CT})$ and outputs 1 if $X' = X$. Otherwise, it outputs 0.

SSS-NIZKPoK.SimSetup$(1^\lambda, \widetilde{X}^*)$: The simulator performs $(\text{MPK}, \text{MSK}) \leftarrow$ FSC.Setup(1^λ). Next it computes a signing key $\text{SK}(f) \leftarrow$ FSC.SKeyGen$(\text{MPK}, \text{MSK}, f)$ for the signing function $f \in \mathbb{F}$ and a decryption key $\text{DK}(g) \leftarrow$ FSC.DKeyGen$(\text{MPK}, \text{MSK}, g)$ for the decryption function $g \in \mathbb{G}$ defined in equation (5) where $\widetilde{X}^*$ will play the role of X^*. It also computes a signing key $\text{SK}(\widetilde{f}) \leftarrow$ FSC.SKeyGen$(\text{MPK}, \text{MSK}, \widetilde{f})$ for the following signing function $\widetilde{f} \in \mathbb{F}$.

$$\widetilde{f}(X\|W\|\beta) = \begin{cases} X\|W\|\beta, \text{ if } [(X,W) \in R \ \bigwedge \ \beta = 1] \ \bigvee \ [X = \widetilde{X}^* \ \bigwedge \ W = 0^\rho \ \bigwedge \ \beta = 0] \\ \perp, \qquad \text{otheriwse} \end{cases} \tag{6}$$

It outputs the simulated common reference string as $\text{CRS} = (\text{MPK}, \text{SK}(f), \text{DK}(g))$ and its simulation trapdoor is $\text{TR} = \text{SK}(\widetilde{f})$.

SSS-NIZKPoK.SimProve$(\text{CRS}, \text{TR}, \widetilde{X}^*)$: The simulator computes the ciphertext for FSC, $\widetilde{\text{CT}} \leftarrow$ FSC.Signcrypt$(\text{MPK}, \text{SK}(\widetilde{f}), \widetilde{X}^*\|0^\rho\|0)$ and outputs the simulated proof as $\widetilde{\pi} = \widetilde{\text{CT}}$.

SSS-NIZKPoK.ExtSetup(1^λ): The extractor generates $(\text{MPK}, \text{MSK}) \leftarrow$ FSC.Setup(1^λ). It identifies some fixed statement $X^* \in \mathbb{L}$ and computes the signing key $\text{SK}(f)$ and decryption key $\text{DK}(g)$ respectively for functions $f \in \mathbb{F}$ and $g \in \mathbb{G}$ defined in equation (5). It additionally computes a decryption key $\text{DK}(g') \leftarrow$ FSC.DKeyGen$(\text{MPK}, \text{MSK}, g')$ for the function $g' \in \mathbb{G}$ where $g' : \{0,1\}^n \rightarrow \{0,1\}^{\rho+1}$ is defined by

$$g'(X\|W\|\beta) = W\|\beta, \text{ for } X\|W\|\beta \in \{0,1\}^n. \tag{7}$$

It outputs the common reference string $\text{CRS} = (\text{MPK}, \text{SK}(f), \text{DK}(g))$ and its extraction trapdoor is $\widehat{\text{TR}} = \text{DK}(g')$.

SSS-NIZKPoK.Extr(CRS, $\widehat{\text{TR}}$, $X, \pi = \text{CT}$): The extractor runs FSC.Unsigncrypt(MPK, DK(g'), CT). If it obtains $W\|1 \in \{0,1\}^{\rho+1}$, then it outputs W. Otherwise, it outputs $\perp$ indicating failure.

We now exhibit that the above construction satisfies all the requirements of an SSS-NIZKPoK system if FSC is a selectively secure functional signcryption scheme.

Theorem 5 (Security of SSS-NIZKPoK). *Assuming that* FSC *is selective message confidential against* CPA *and selective ciphertext unforgeable against* CMA *as per the definition of §3, the* SSS-NIZKPoK *system described above satisfies all the criteria of* SSS-NIZKPoK *defined in §2.2.*

Proof. The proofs of the different properties satisfied by the above SSS-NIZKPoK scheme based on different features of FSC is provided below:

- **Perfect Completeness**: The perfect completeness of SSS-NIZKPoK is immediate from the correctness of FSC.

- **Statistical Soundness**: Statistical soundness of SSS–NIZKPoK follows from the fact that the only input $X\|W\|\beta$ with (X, W) not necessarily in R for which g outputs X is the string $X^*\|0^\rho\|0$ and in SSS-NIZKPoK.Setup X^* is taken to be a member of $\mathbb{L}$.

- **Computational Zero-Knowledge**: We argue that if there is an adversary $\mathcal{A}$ that breaks the computational zero-knowledge property of SSS-NIZKPoK, then there is an adversary $\mathcal{C}$ that breaks the selective CPA-message confidentiality of FSC using $\mathcal{A}$ as a subroutine. $\mathcal{C}$ interacts with $\mathcal{A}$ as follows:

  - $\mathcal{C}$ begins by initializing $\mathcal{A}$ and receiving $(\widetilde{X}^*, W^*) \in R$ from $\mathcal{A}$. $\mathcal{C}$ submits $(f_0^*, z_0^*), (f_1^*, z_1^*)$ to its *message confidentiality challenger* $\mathcal{B}$ for FSC where $f_0^* = f, f_1^* = \widetilde{f}$ constructed from $\widetilde{X}^*$ as defined in equations (5), (6) respectively, and $z_0^* = \widetilde{X}^*\|W^*\|1, z_1^* = \widetilde{X}^*\|0^\rho\|0$.

  - $\mathcal{C}$ gets the public parameters MPK for FSC from $\mathcal{B}$.

  - $\mathcal{C}$ also queries $\mathcal{B}$ with the signing function $f \in \mathbb{F}$ and the decryption function $g \in \mathbb{G}$ constructed using $\widetilde{X}^*$ as in equation (5). Note that $g(f_0^*(z_0^*)) = \widetilde{X}^* = g(f_1^*(z_1^*))$ as $(\widetilde{X}^*, W^*) \in R$. Thus g satisfies the restriction on the decryption key query in the selective CPA-message confidentiality game for FSC. $\mathcal{B}$ returns the signing key SK(f) and the decryption key DK(g) to $\mathcal{C}$. $\mathcal{C}$ sets the common reference string as CRS = (MPK, SK(f), DK(g)).

  - $\mathcal{C}$ also receives a challenge ciphertext CT* from $\mathcal{B}$ and designates it as the proof π^* on $\widetilde{X}^*$.

  - $\mathcal{C}$ hands (CRS, π^*) to $\mathcal{A}$.

  - Finally, $\mathcal{A}$ outputs a bit $\widetilde{b} \in \{0,1\}$. $\mathcal{C}$ also outputs $\widetilde{b}$.

  Clearly, if $\mathcal{B}$ gave CT$^* \leftarrow$ FSC.Signcrypt(MPK, FSC.SKeyGen(MPK, MSK, f_0^*), z_0^*), then (CRS, π^*) corresponds to the "real" scenario. On the other hand, if $\mathcal{B}$ gave CT$^* \leftarrow$ FSC.Signcrypt(MPK, FSC.SKeyGen(MPK, MSK, f_1^*), z_1^*), then (CRS, π^*) corresponds to the "simulation" scenario. Hence, if $\mathcal{A}$ breaks the computational zero-knowledge property of SSS-NIZKPoK, then $\mathcal{C}$ breaks the selective CPA-message confidentiality of FSC.

- **Knowledge Extraction**: Obviously, the CRS values generated by SSS-NIZKPoK.Setup and SSS-NIZKPoK.ExtSetup are identically distributed. Now we will argue that the probability

that the extraction will fail is the same as the probability of forging a signcryption in FSC. Consider a PPT adversary $\mathcal{A}$ that wins the extraction game with non-negligible advantage. We construct a PPT adversary $\mathcal{C}$ that forges a signcryption of FSC with non-negligible advantage using $\mathcal{A}$ as a sub-routine. $\mathcal{C}$ interacts with $\mathcal{A}$ as follows:

- $\mathcal{C}$ identifies some statement $X^* \in \mathbb{L}$ and submits the message $X^* \| 0^\rho \| 0$ to its *unforgeability challenger* $\mathcal{B}$ for FSC as the challenge message on which it wants to forge a signcryption.
- $\mathcal{C}$ receives the public parameters MPK for FSC from $\mathcal{B}$.
- $\mathcal{C}$ constructs the function $f \in \mathbb{F}$ defined in equation (5) and queries a signing key for f to $\mathcal{B}$. Note that $X^* \| 0^\rho \| 0$ is outside the range of f. Thus, f satisfies the restriction on the signing key query in the selective CMA-ciphertext unforgeability game for FSC. $\mathcal{B}$ returns the signing key $\mathrm{SK}(f)$ to $\mathcal{C}$.
- $\mathcal{C}$ also queries $\mathcal{B}$ with the decryption functions $g \in \mathbb{G}$ constructed using X^* as in equation (5) and receives back the decryption key $\mathrm{DK}(g)$.
- $\mathcal{C}$ gives the common reference string $\mathrm{CRS} = (\mathrm{MPK}, \mathrm{SK}(f), \mathrm{DK}(g))$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs a statement-proof pair $(\widetilde{X}^*, \pi^* = \mathrm{CT}^*)$. $\mathcal{C}$ outputs CT^* as a forgery on $X^* \| 0^\rho \| 0$.

Clearly, the simulation is perfect. Note that if $(\widetilde{X}^*, \pi^*)$ is a valid statement-proof pair, then it must hold that $\widetilde{X}^* \leftarrow \mathsf{FSC.Unsigncrypt}(\mathrm{MPK}, \mathrm{DK}(g), \mathrm{CT}^*)$. Then, from the definition of g, it follows that either (I) the message that has been signcrypted within CT^* is $\widetilde{X}^* \| \widetilde{W}^* \| 1$ for some $\widetilde{W}^*$ such that $(\widetilde{X}^*, \widetilde{W}^*) \in R$ or (II) $\widetilde{X}^* = X^*$ and the message signcrypted within CT^* is $X^* \| 0^\rho \| 0$. Now recall that the extraction trapdoor used in the extraction process is $\widehat{\mathrm{TR}} = \mathrm{DK}(g')$ for the function $g' \in \mathbb{G}$ defined in equation (7). From the definition of g', it is clear that, in case (I), $\mathsf{FSC.Unsigncrypt}(\mathrm{MPK}, \mathrm{DK}(g'), \mathrm{CT}^*)$ gives $\widetilde{W}^* \| 1$, so that in that case the extraction process succeeds and $\mathcal{A}$ cannot win the above game. Thus if $\mathcal{A}$ wins the above game, then case(II) must hold, i.e., the signcrypted message in CT^* must be $X^* \| 0^\rho \| 0$, so that $\mathsf{FSC.Unsigncrypt}(\mathrm{MPK}, \mathrm{DK}(g'), \mathrm{CT}^*)$ outputs $0^\rho \| 0$ and the extraction process fails. Thus if $\mathcal{A}$ wins in the above game, then CT^* must be a valid forgery on $X^* \| 0^\rho \| 0$ and hence, $\mathcal{C}$ wins in its unforgeability game.

- **Statistical Simulation Soundness**: Note that according to the description of the algorithms SSS-NIZKPoK.SimSetup and SSS-NIZKPoK.SimProve, for any statement $\widetilde{X}^*$, if the simulated common reference string and proof are given to the adversary, the only statement for which the adversary can output a false proof that passes the verification is $\widetilde{X}^*$. This is because, the simulated common reference string would contain the decryption key $\mathrm{DK}(g)$ for the function $g \in \mathbb{G}$ constructed using $\widetilde{X}^*$ as in equation (5). Therefore, the unsigncryption algorithm executed during the verification process would output the statement only if either the proof is computed using a valid statement-witness pair or the statement on which the proof is computed is $\widetilde{X}^*$ itself. Thus, even after seeing simulated proofs of arbitrary statement the adversary cannot produce a convincing proof on a false statement.

□

6.2 IO from FSC

Note that from any selectively secure FSC scheme we can obtain a *selectively secure functional encryption* (FE) scheme (for background on FE see [GGH+13b]) with the same message space and decryption function family as the underlying FSC scheme by including a signing key in

the public parameters of FE for the signing function which is simply the *identity function* on the message space of FSC. Recently, Ananth et al. [AJS15] has shown how to construct IO for P/poly from selectively secure FE. Following these, we can design an IO for P/poly from FSC. The details are omitted.

7 Conclusion

In this paper, we described a *new* cryptographic primitive called *functional signcryption* (FSC) which is a blend of FE and FS and provides a more efficient solution for controlling the signing and decryption rights in a multi-user confidential and authenticated digital communication or storage system. We also presented a *generic construction* of FSC supporting signing and decryption functions representable as *polynomial-size circuits* utilizing existing cryptographic building blocks, namely, IO for polynomial-size circuits and SSS-NIZKPoK system for NP relations. As application of this ambitious primitive, we constructed the *first* ABSC scheme that supports *general polynomial-size circuits* and showed that FSC can also be employed to build SSS-NIZKPoK for NP relations and IO for polynomial-size circuits.

There are a number of open research directions pertaining to FSC. Firstly, in view of making FSC more practicable one may attempt to construct FSC, possibly for restricted classes of functions, from weaker and more efficient primitives rather than using IO or SSS-NIZKPoK. Secondly, it would be quite interesting to develop an FSC scheme that provides adaptive security as opposed to our selectively secure construction. Thirdly, in this work, we did not consider the simulation paradigm to define security for FSC. Formulating a simulation-based security notion for FSC and identifying the possibilities and impossibilities of that simulation-based security definition have important theoretical significance. A fourth fascinating line of research would be to develop a meaningful notion of *function privacy* in the context of FSC and find out its importance in practical scenarios. Also, it is instructive to investigate whether our non-function-private FSC construction can be extended to achieve that new definition of function privacy for FSC applying a similar transformation technique as has been employed in [BS15] in case of FE. Finally, we believe that FSC can be utilized as a tool for constructing many more fundamental cryptographic primitives and discovering those new applications of FSC is another interesting area of research.

References

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. *IACR Cryptology ePrint Archive*, 2013:689, 2013.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Achieving compactness generically: Indistinguishability obfuscation from non-compact functional encryption. *IACR Cryptology ePrint Archive*, 2015:730, 2015.

[Bar86] David A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc 1. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 1–5. ACM, 1986.

[BF14] Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. In *Public-Key Cryptography–PKC 2014*, pages 520–537. Springer, 2014.

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

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM, 2012.

[BMS13] Michael Backes, Sebastian Meiser, and Dominique Schröder. Delegatable functional signatures. *IACR Cryptology ePrint Archive*, 2013:408, 2013.

[BS15] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. In *Theory of Cryptography*, pages 306–324. Springer, 2015.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography*, pages 253–273. Springer, 2011.

[CLT15] Jean-Sebastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. *IACR Cryptology ePrint Archive*, 2015:162, 2015.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Advances in Cryptology-EUROCRYPT 2013*, volume 7881, pages 1–17. Springer, 2013.

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

[GGH+13c] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *Advances in Cryptology–CRYPTO 2013*, pages 479–499. Springer, 2013.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. *IACR Cryptology ePrint Archive*, 2014:666, 2014.

[GLSW14] Craig Gentry, Allison B Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. *IACR Cryptology ePrint Archive*, 2014:309, 2014.

[GNSN10] Martin Gagné, Shivaramakrishnan Narayan, and Reihaneh Safavi-Naini. Threshold attribute-based signcryption. In *Security and Cryptography for Networks*, pages 154–171. Springer, 2010.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *Advances in Cryptology-EUROCRYPT 2006*, pages 339–358. Springer, 2006.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM* (*JACM*), 59(3):11, 2012.

[Gro06] Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *Advances in Cryptology–ASIACRYPT 2006*, pages 444–459. Springer, 2006.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography* (*Chapman & Hall/Crc Cryptography and Network Security Series*). Chapman & Hall/CRC, 2007.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437. ACM, 1990.

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

[RD14a] Y Sreenivasa Rao and Ratna Dutta. Expressive attribute based signcryption with constant-size ciphertext. In *Progress in Cryptology–AFRICACRYPT 2014*, pages 398–419. Springer, 2014.

[RD14b] Y Sreenivasa Rao and Ratna Dutta. Expressive bandwidth-efficient attribute based signature and signcryption in standard model. In *Information Security and Privacy*, pages 209–225. Springer, 2014.

[Wat14] Brent Waters. A punctured programming approach to adaptively secure functional encryption. *IACR Cryptology ePrint Archive*, 2014:588, 2014.

[WH11] Changji Wang and Jiasen Huang. Attribute-based signcryption with ciphertext-policy and claim-predicate mechanism. In *Computational Intelligence and Security* (*CIS*)*, 2011 Seventh International Conference on*, pages 905–909. IEEE, 2011.

[Zhe97] Yuliang Zheng. Digital signcryption or how to achieve cost (signature & encryption)ń cost (signature)+ cost (encryption). In *Advances in Cryptology–CRYPTO'97*, pages 165–179. Springer, 1997.