IET Research Journals

Submission Template for IET Research Journal Papers

IET Journals
The Institution of Engineering and Technology

Towards Stronger Functional Signatures

ISSN 1751-8644
doi: 0000000000
www.ietdl.org

Carlo Brunetta[1*], *Bei Liang*[1], *Aikaterini Mitrokotsa*[1]

[1] *Chalmers University of Technology, Gothenburg, Sweden*
* *E-mail: brunetta@chalmers.se*

Abstract: *Functional digital Signatures* (FS) schemes introduced by Boyle, Goldwasser and Ivan (PKC 2014) provide a method to generate fine-grained digital signatures in which a master key-pair $(\mathsf{msk}, \mathsf{mvk})$ is used to generate a signing secret-key sk_f for a function f that allows to sign any message m into the message $f(\mathsf{m})$ and signature σ. The verification algorithm takes the master verification-key mvk and checks that the signature σ corresponding to $f(\mathsf{m})$ is valid. In this paper, we enhance the FS primitive by introducing a function public-key pk_f that acts as a commitment for the specific signing key sk_f. This public-key is used during the verification phase and guarantees that the message-signature pair is indeed the result generated by employing the specific key sk_f in the signature phase, a property not achieved by the original FS scheme. This enhanced FS scheme is defined as *Strong Functional Signatures* (SFS) for which we define the properties of unforgeability as well as the function hiding property. Finally, we provide an unforgeable, function hiding SFS instance in the random oracle model based on Boneh-Lynn-Shacham signature scheme (ASIACRYPT 2001) and Fiore-Gennaro's publicly verifiable computation scheme (CCS 2012).

Keywords: Functional Signatures, Verifiable Computation, Function Privacy

1 Introduction

Digital signatures, introduced by Diffie and Hellman [7], is a valuable cryptographic primitive that provides important integrity guarantees, *i.e.,* a signed message allows the receiver to verify that the message was indeed signed by the claimed signer. *Functional digital signatures* (FS), introduced by Boyle, Goldwasser and Ivan [6] as a general extension of classic digital signatures [12], allow generating signatures in a more *fine-grained manner*; thus, being very useful in multiple applications, *e.g.*, scenarios where the *delegation of signing rights* has to be considered. Functional digital signatures require a *trusted authority* to hold a *master secret key*. Given a description of a function f, the authority, using the master secret key, can generate a limited functional signing key sk_f associated with the function f. Anyone that has access to the signing key sk_f and a message m, can compute $f(\mathsf{m})$ **and** the corresponding *functional signature* σ of $f(\mathsf{m})$.

Let us employ an example related to photo-processing given by Boyle *et al.* [6] to explain how FS works. When performing photo-processing, a digital camera is required to produce signed photos. One may want to allow photo-processing software to perform minor touch-ups of the photos, such as changing the contrast, but not allow more significant changes such as merging two photos or cropping a photo. Boyle *et al.*argued that FS could be used in such a setting to provide the photo processing software with a restricted key, which enables it to sign only specific modifications of an original photo. Let us assume there are three different pictures partitioned into three areas and coloured in red, blue and yellow but in different order, as represented in Figure 1.

The functionality of f_1 is to exchange the colour of areas 2 and 3, while f_2 is used to exchange the colour of areas 1 and 3, and f_3 to exchange the colour of areas 1 and 2. Using the secret key sk_{f_1} to sign the photo ϕ_1, we obtain the signed new photo y_1. With the restricted keys sk_{f_2} and sk_{f_3}, we can obtain two signed photos with the same picture on it, namely y_2 and y_3. Using functional signatures, given y_1, y_2 and y_3, the appreciator (not the one who provides the original picture) only knows they are three certified photos.

Generally, if we consider two functions f and g and two messages m, m' such that $f(\mathsf{m}) = g(\mathsf{m}') = y$, then, given y and the corresponding functional signature σ, FS cannot be used to certify that the function value y is indeed computed from the queried function f and m rather than from g and m'. The latter yields from the *function privacy* property of FS [6], namely given y and σ, any adversary is unable to tell which function f or g was used to compute the value y even when given both functional signing keys sk_f and sk_g.

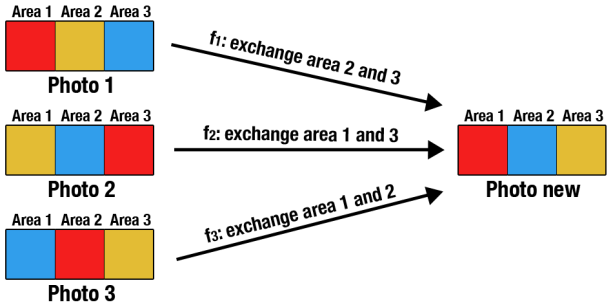

Fig. 1: An illustrated example of collisions from different messages and functions in a *functional signature* scheme.

What if we wish to make the appreciator classify that a signed photo y, is indeed the outcome of applying an "allowed" *function without revealing* "which" *one?*

Our idea: to allow an appreciator/verifier to distinguish between the usage of different secret keys, *e.g.* sk_f and sk_g, we introduce a *function public key*, *i.e.* pk_f and pk_g, that is just used in the verification phase. The public key pk_f can be seen as a commitment for the specific and related secret key sk_f allowing to distinguish between the evaluation and signatures $(f(\mathsf{m}), \sigma_1)$ and $(g(\mathsf{m}'), \sigma_2)$ even in the case that $f(\mathsf{m}) = g(\mathsf{m}')$. This *"key-addition"* directly affects the FS function privacy property that changes from *"the verifier cannot retrieve which function was computed"* to the stronger concept of *"the verifier cannot retrieve which function was computed* ***despite*** *knowing the related public key"*. We capture this idea into the enhanced definition of *Strong Functional Signature (SFS)*, an *Functional Signature (FS)*-like scheme with function public keys that allows the verification of function evaluations' signatures **and** guarantees the correct function evaluation **while** maintaining the function hidden.

Example - Computational Authorisation for Cloud Computing: our SFS primitive could be used in the example previously described, as well as in more general applications related to the cloud-assisted

setting which are alike to the certification authorities' infrastructure **but** for function application and not only for identity authentication.

As depicted in Fig. 2, let us consider a cloud service $\mathcal{T}$ that offers to service providers $\mathcal{S}_i$ the possibilities to *register* their functionalities f_i in exchange of guaranteeing function hiding and the correct authentication whenever a user $\mathcal{U}_j$ wants to verify the authenticity and correctness of the output of such hidden functionalities. In other words, $\mathcal{S}_i$ will register the function f_i, obtain sk_{f_i} from $\mathcal{T}$ and, at the same time, $\mathcal{T}$ will publish the public key pk_{f_1} with some application label, *e.g.* it might be published into an *"Authorised"* functionality list. Later on, the user $\mathcal{U}_j$ requires $\mathcal{S}_i$ to process their data, obtains the output y with signature σ and wants to verify that y is indeed *correctly computed by an authorised* function. Therefore, $\mathcal{U}_j$ obtains the list of authorised public keys pk_{f_i} and verifies that (y, σ) is valid by finding a public key pk_f that pass the SFS validation algorithm. Additionally, $\mathcal{U}_j$ is unable to infer the precise function f from the public key pk_f thus the cloud service $\mathcal{T}$ guarantees to the service provider $\mathcal{S}$ that the function is kept private.

Observe that the cloud service $\mathcal{T}$ has the power to modify the status of the public keys, *e.g.* a public key pk_g might be completely *"revoked"* by removing it from all the public key's lists.

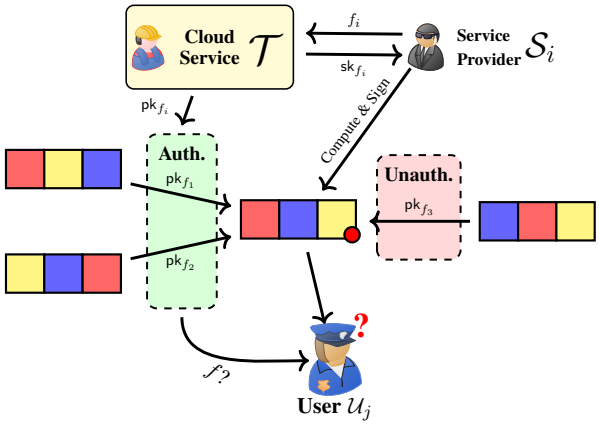

Fig. 2: Strong functional signatures in the cloud computational authentication scenario.

It is obvious that FS [1, 2, 6] does not have the features of checking if the outcome is resulted from the authorised functions, neither achieves this concept of *"revocability"*. In fact, in FS, since only mvk is required to verify the validity of (y, σ), it is not possible to check if a specific function was applied to output y, while our SFS make it possible by providing restricted public keys *w.r.t.* each function, which are employed in the verification process.

Moreover, in traditional FS schemes, it is indeed impossible to *"revoke"* a specific signing key, since the verification process would always work. However, in our introduced SFS notion, by incorporating the public keys in the verification process, we are able to revoke the signing capability for a restricted signing key thus allowing the trusted third party that owns the master key pair, to create a more fine-grained control over the generated function key pairs.

Our Results: our results can be summarised as follows:

- we formally define the notion of SFS with unforgeability and function hiding properties;
- we provide a variation of Boneh *et al.*'s BLS signature scheme [5] and a variation of Fiore and Gennaro's verifiable computation scheme [8]. We prove that the Fiore and Gennaro's VC scheme satisfies the *Public Verifiable Computation (PVC)* privacy properties;
- based on our variations, we give an instantiation in the random oracle model of an SFS scheme for the polynomial function family which is adaptively unforgeable and satisfies the function hiding property.

The starting point of our instantiation of SFS is to use the BLS signature scheme [5] in combination with the Fiore-Gennaro's publicly VC scheme [8] that is compatible with the algebraic structure and assumptions of the BLS signatures. We denote with $\overline{\mathsf{BLS}}$ the variation of the BLS signature and with $\overline{\mathsf{VC}}$ the variation of the Fiore-Gennaro's VC scheme, that we propose. The design-trick behind our instantiation is to *create* a *master key-pair* as an algebraic one-way instance and use it as a *"transposition"* for the secret key of the schemes, *e.g.* $\overline{\mathsf{BLS}}.\mathsf{Setup}(\lambda) \rightarrow (\mathsf{MSK}, \mathsf{MPK})$ is equal to $(\beta, e(g_1, g_2)^{\beta})$ for some $\beta \in \mathbb{Z}_p$ and whenever we sample a fresh secret value $\alpha \in \mathbb{Z}_p$ in order to compute the $\overline{\mathsf{BLS}}$ and the $\overline{\mathsf{VC}}$ keys, we just consider the new secret $\alpha + \beta$ obtained by translating α by β. Thus, all the evaluation/secret-keys are computed as if $\alpha+\beta$ is the randomness sampled while the verification/public-keys are published as *"local keys"*, *e.g.* we publish $e(g_1, g_2)^{\alpha}$ and not $e(g_1, g_2)^{\alpha+\beta}$. In this way, the two variated schemes become *"entangled"* thus implying a stricter relation during execution and verification. In a nutshell, the SFS instantiation combines the two schemes such that the verifiable computation $\overline{\mathsf{VC}}$ computes the secret function and provide the proof of correct computation while the signature scheme $\overline{\mathsf{BLS}}$ is used to sign the result and forcedly relate it to the $\overline{\mathsf{VC}}$ results.

Related Work: SFS are inspired by Boyle *et al.* [6] FS construction and are closely related to *Signatures of Correct Computation (SCC)* proposed by Papamanthou, Shi and Tamassia [13] as well as PVC proposed by Parno *et al.* [14] and Fiore and Gennaro [8].

Functional Signatures. This work is inspired by the notion of *Functional Signatures* (FS) introduced by Boyle *et al.* [6]. They firstly proposed the formal definition of FS with *unforgeability security* as well as two additional desirable properties: *function privacy* and *succinctness*. Boyle *et al.* defined FS and gave a construction for an FS scheme, based on one-way functions and satisfying the unforgeability **but not** the succinctness or function privacy properties. Furthermore, they showed how to convert any FS without the function privacy or succinctness properties into an FS scheme that is succinct and function-private by using a SNARK scheme [3, 4, 11]. They also showed how to use an FS scheme to construct a delegation scheme [10], *i.e.*, non-interactive verifiable computation.

Signatures of Correct Computation. Papamanthou, Shi and Tamassia introduced *Signatures of Correct Computation (SCC)* for verifying the correctness of a computation outsourced in the cloud [13]. In the SCC model, an authority wishes to outsource the execution of a function f to an untrusted server. It generates a pair of master keys along with a verification key $\mathsf{FK}(f)$ for that function which will be used during verification. Note that the existence of such a *verification key* for a function f and the requirement of being used for verification are similar to our formulation of SFS. The server can then return a signature σ on a value y, which certifies that the result y is indeed the correct outcome of the function f evaluated on some input. In the syntax of SCC [13], anyone with the public verification key can verify that an untrusted server correctly computed a function f on a specific input m. However, the verification algorithm requires the specific input m, used to compute $f(\mathsf{m})$, to be taken as input, which means that only the client or someone who knows the input m can verify the correctness of the computation. Therefore, SCC would not achieve any privacy with respect to the input m. In contrast, our SFS allows anyone to perform the verification without knowledge of the specific input m.

Publicly Verifiable Computation. Parno *et al.* [14] have proposed a *publicly verifiable computation (PVC)* in which they consider a PVC scheme achieving two desirable properties: *public delegatability* and *public verifiability*. Their definition of PVC includes a $\mathsf{ProbGen}$ algorithm, which encodes a user's inputs m to a server's inputs σ_{m} and simultaneously prepares an element ρ_{m} to be used for verification. Thus, ρ_{m} can be used to publicly verify that the server returned a correct value. The *public delegation* property refers to the existence of a *public delegation key* pk_f *for the function* f, *i.e.*, the key used

in the ProbGen algorithm, and publicly available to anyone. Thus, anyone can use the key and delegate the computation to the cloud.

Parno *et al.* [14] also gave a construction of a VC scheme with public delegation and public verifiability from any *Attribute-Based Encryption (ABE)*, which is unfortunately not appropriate to be employed in order to instantiate a SFS since additional transformations are needed.

Another closely related work is the one by Fiore and Gennaro [8], who presented a very efficient PVC scheme tailored for multivariate polynomials over a finite field based on bilinear maps. We present a variation of their VC scheme by introducing a separate Setup algorithm to generate a master key pair for the scheme so that the keys for the evaluation of different functions could be executed multiple times using the same parameters for the scheme, which allows the evaluation of multiple functions on the same instance produced by ProbGen.

Paper organisation. In Section 2, we describe the notations and review the primitives used in the paper. In Section 3, we propose two variances: one of Boneh *et al.*'s signature scheme, denoted $\overline{\mathsf{BLS}}$, and one of the Fiore-Gennaro's PVC scheme, denoted $\overline{\mathsf{VC}}$. In Section 4, we provide the definition of SFS and its security properties and we instantiate an unforgeable and function hiding SFS using the $\overline{\mathsf{BLS}}$ and the $\overline{\mathsf{VC}}$ schemes.

2 Preliminaries

In the following section, we define the notations used through out the paper. We also provide the assumptions and the definitions of the building blocks that our constructions rely on.

2.1 Notations and Assumptions

In the paper, we denote with $x \leftarrow\!\!\$\ X$ the random uniform sampling in the set X, with λ the security parameter. We denote with $\vec{v}$ a vector and with $\mathbb{Z}_p$ the ring with p elements. When not specified, p always represents either a prime or a power of it. Let $\mathsf{Pr}(E)$ denote the probability that the event E occurs. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups of the same order with generators g_1, g_2, g_T correspondingly and the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ of type-3, i.e. there does **not** exists an efficient homomorphism map $\psi : \mathbb{G}_2 \to \mathbb{G}_1$.

Definition 1 (co-Computation Diffie Hellman [5, 8]). *Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups of prime order p. Let $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ be generators and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ bilinear map of type-3, i.e. there does not exists an efficient homomorphism map $\psi : \mathbb{G}_2 \to \mathbb{G}_1$. We sample uniformly at random $a, b \leftarrow\!\!\$\ \mathbb{Z}_p$ and define the advantage of an adversary $\mathcal{A}$ in solving the **co-Computational Diffie Hellman** (co-CDH) problem as*

$$\mathsf{Adv}^{\textit{co-CDH}}_{\mathcal{A}}(\lambda) = \mathsf{Pr}(\mathcal{A}(p, g_1, g_2, g_1^a, g_2^b) = g_1^{ab})$$

If for all adversaries $\mathcal{A}$ it exists a negligible ϵ such that $\mathsf{Adv}^{\textit{co-CDH}}_{\mathcal{A}}(\lambda) \leq \epsilon$, then the co-CDH Assumption *ϵ-holds for the groups $\mathbb{G}_1, \mathbb{G}_2$.*

2.2 Closed Form Efficient PRFs

A closed form efficient PRF (*Closed Form Efficient (CFE)-PseudoRandom Function (PRF)*), defined by Fiore and Gennaro [8] consists of three algorithms CF.KeyGen, CF.H and CF.Eval. CF.KeyGen takes as input a security parameter λ and outputs a secret key K, from the key space $\mathcal{K}$, and some public parameters pp that specify the domain $\mathcal{X}$ and range $\mathcal{Y}$ of the function. For a fixed secret key K, $\mathsf{CF.H}_K$ takes as input a value $x \in \mathcal{X}$ and outputs a value $y \in \mathcal{Y}$. It satisfies the *pseudo-randomness* property: for every PPT adversary $\mathcal{A}$, $(K, \mathsf{pp}) \leftarrow \mathsf{CF.KeyGen}(\lambda)$ and any random function $\xi : \mathcal{X} \to \mathcal{Y}$:

$$\epsilon_{\mathsf{PRF}} = \left| \begin{array}{r} \mathsf{Pr}\Big(\mathcal{A}^{\mathsf{CF.H}_K(\cdot)}(\lambda, \mathsf{pp}) = 1\Big) - \\ - \mathsf{Pr}\Big(\mathcal{A}^{\xi(\cdot)}(\lambda, \mathsf{pp}) = 1\Big) \end{array} \right| \leq \mathsf{negl}(\lambda)$$

Additionally, the scheme is required to achieve *closed form efficiency*: consider a generic computation ϕ that has as input l random values $R_1, \ldots, R_l \in \mathcal{Y}$ and a vector of m arbitrary values $\vec{x} = (x_1, \ldots, x_m)$. Assume that the fastest computation time that takes to compute $\phi(R_1, \ldots, R_l, x_1, \ldots, x_m)$ is T. Let $\vec{z} = (z_1, \ldots, z_l)$ be a l-tuple of arbitrary values in the domain $\mathcal{X}$. The CF.PRF is said to achieve *closed form efficiency* for $(\phi, \vec{z})$ if the algorithm CF.Eval has running time $o(T)$ and it holds

$$\mathsf{CF.Eval}_{(\phi,\vec{z})}(K, \vec{x}) = \phi(\mathsf{CF.H}_K(z_1), \ldots, \mathsf{CF.H}_K(z_l), x_1, \ldots, x_m)$$

Fiore and Gennaro [8] give constructions of closed form efficient PRFs for multivariate polynomials and matrix multiplication, based on the decision linear assumption.

2.3 Functional Signatures

Boyle *et al.* [6] introduced *functional digital signatures* (FS), a cryptographic primitive that can be employed to achieve *signing delegation*.

Definition 2 (Functional Signature [6]). *A Functional Signature scheme for a message space $\mathcal{M}$ and function family $\mathcal{F} = \{f : \mathcal{D}_f \to \mathcal{M}\}$ consists of the PPT algorithms $\mathsf{FS} = (\mathsf{FS.Setup}, \mathsf{FS.KeyGenFS.Sign}, \mathsf{FS.Verify})$ defined as:*

- *$\mathsf{FS.Setup}(\lambda) \to (\overline{\mathsf{msk}}, \overline{\mathsf{mvk}})$: the setup algorithm takes as input the security parameter λ and outputs the master signing key $\overline{\mathsf{msk}}$ and the master verification key $\overline{\mathsf{mvk}}$.*
- *$\mathsf{FS.KeyGen}(\overline{\mathsf{msk}}, f) \to \overline{\mathsf{sk}}_f$: the key generation algorithm takes as input the master signing key and a function $f \in \mathcal{F}$ and outputs a signing key $\overline{\mathsf{sk}}_f$.*
- *$\mathsf{FS.Sign}(f, \overline{\mathsf{sk}}_f, m) \to (f(m), \overline{\sigma})$: the signing algorithm takes as input the signing key for a function f and an input $m \in \mathcal{D}_f$, and outputs $f(m)$ and a signature $\overline{\sigma}$ of $f(m)$.*
- *$\mathsf{FS.Verify}(\overline{\mathsf{mvk}}, m', \overline{\sigma}) \to \{0, 1\}$: the verification algorithm takes as input the master verification key $\overline{\mathsf{mvk}}$, a message m' and a signature $\overline{\sigma}$, and outputs 1 if the signature is valid.*

The definition requires the following conditions to hold:

Correctness: a *Functional Signature (FS)* scheme is *correct* if for all functions $f \in \mathcal{F}$, messages $\mathsf{m} \in \mathcal{D}_f$, $(\overline{\mathsf{msk}}, \overline{\mathsf{mvk}})$ obtained from $\mathsf{FS.Setup}(\lambda)$, $\overline{\mathsf{sk}}_f$ obtained from $\mathsf{FS.KeyGen}(\overline{\mathsf{msk}}, f)$ and $(\mathsf{m}', \overline{\sigma})$ obtained from $\mathsf{FS.Sign}(f, \overline{\mathsf{sk}}_f, \mathsf{m})$, it holds that $\mathsf{FS.Verify}(\overline{\mathsf{mvk}}, \mathsf{m}', \sigma) = 1$.

Succinctness: there exists a polynomial $s(\cdot)$ such that for every $\lambda \in \mathbb{N}$, function $f \in \mathcal{F}$, message $\mathsf{m} \in \mathcal{D}_f$, master keys $(\overline{\mathsf{msk}}, \overline{\mathsf{mvk}}) \leftarrow \mathsf{FS.Setup}(\lambda)$, function key $\overline{\mathsf{sk}}_f$ obtained from $\mathsf{FS.KeyGen}(\overline{\mathsf{msk}}, f)$, and $(f(\mathsf{m}), \overline{\sigma}) \leftarrow \mathsf{FS.Sign}(\overline{\mathsf{sk}}_f, \mathsf{m})$, it holds with probability 1 that $|\overline{\sigma}| \leq s(\lambda, |f(\mathsf{m})|)$.

Unforgeability: FS is *unforgeable* if the probability of any PPT algorithm $\mathcal{A}$ in the FS unforgeability experiment $\mathsf{Exp}^{\mathsf{FS.UNF}}_{\mathsf{FS}}(\mathcal{A})$, depicted in Figure 3, to output 1 is negligible. Namely,

$$\mathsf{Adv}^{\mathsf{FS.UNF}}_{\mathcal{A},\mathsf{FS}}(\lambda) = \mathsf{Pr}\Big(\mathsf{Exp}^{\mathsf{FS.UNF}}_{\mathsf{FS}}(\mathcal{A}) = 1\Big) \leq \mathsf{negl}(\lambda)$$

Function privacy: FS is *function private* if the advantage of any PPT algorithm $\mathcal{A}$ in the FS function privacy experiment

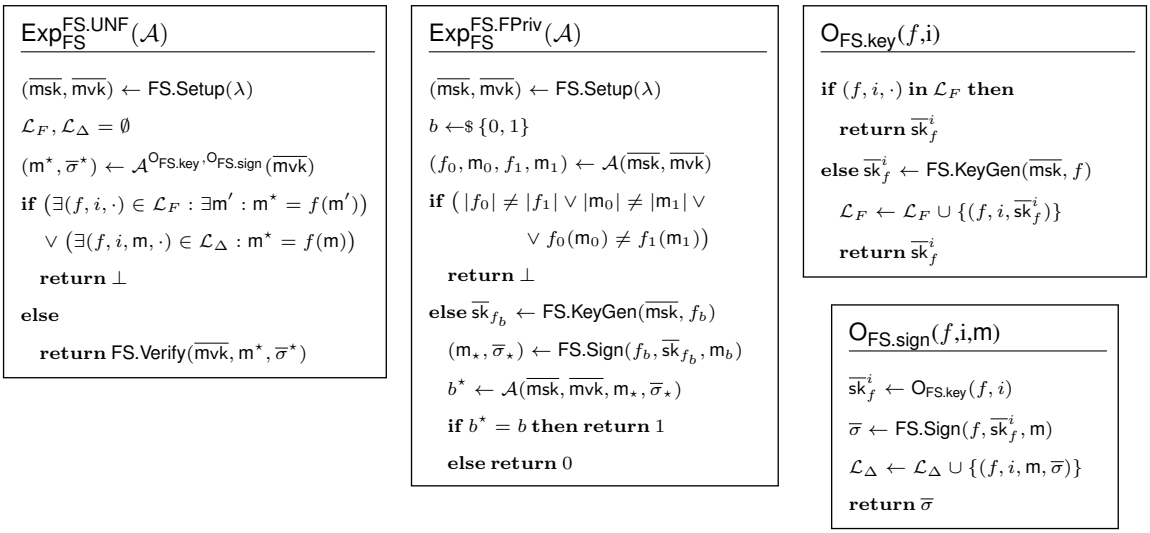

Fig. 3: Functional signature unforgeability and function privacy experiments.

$\mathsf{Exp}^{\mathsf{FS.FPriv}}_{\mathsf{FS}}(\mathcal{A})$, depicted in Figure 3 is negligible. Namely,

$$\mathsf{Adv}^{\mathsf{FS.FPriv}}_{\mathcal{A},\mathsf{FS}}(\lambda) = \left|\Pr\left(\mathsf{Exp}^{\mathsf{FS.FPriv}}_{\mathsf{FS}}(\mathcal{A}) = 1\right) - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$$

2.4 The BLS Signature Scheme

In this section, we will report the Boneh *et al.*'s signature scheme [5]. Let $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map in the security parameter λ. Let $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$ be a full-domain hash function. The BLS signature scheme [5] with the message space $\mathcal{M} = \{0,1\}^*$ comprises of the following three algorithms:

- $\mathsf{BLS.KeyGen}(\lambda) \to (\mathsf{PK}, \mathsf{SK})$: given a security parameter λ, sample a secret value $\mathsf{SK} \leftarrow\$ \mathbb{Z}_p$ and compute as the public key $\mathsf{PK} = g_2^{\mathsf{SK}}$.
- $\mathsf{BLS.Sign}(\mathsf{SK}, \mathsf{m}) \to \ddot{\sigma}$: given a secret key SK and a message $\mathsf{m} \in \mathcal{M}$, compute $\mathsf{H}(\mathsf{m})$ and output the signature $\ddot{\sigma} = \mathsf{H}(\mathsf{m})^{\mathsf{SK}}$.
- $\mathsf{BLS.Verify}(\mathsf{PK}, \mathsf{m}, \ddot{\sigma}) \to \{0,1\}$: given a public key PK, a message m and a signature $\ddot{\sigma}$, check $e(\ddot{\sigma}, g_2) \stackrel{?}{=} e(\mathsf{H}(\mathsf{m}), \mathsf{PK})$ and output 1 if it is true, otherwise output 0.

The BLS scheme is existentially unforgeable against chosen message attacks in the random oracle model (ROM), assuming the co-CDH assumption of Definition 1 holds.

2.5 Verifiable Computation

A *verifiable computation* (VC) scheme allows a client to delegate the computation of a function f to a server so that the client is able to verify the correctness of the result returned by the server with less computation cost than evaluating the function directly. We describe the definition of a *verifiable computation* (VC) scheme introduced by Parno *et al.* [14] and Fiore and Gennaro [8].

Definition 3 (Verifiable Computation [8, 14])**.** *A verifiable computation scheme* VC *is defined by the following algorithms:*

- $\mathsf{VC.KeyGen}(\lambda, f) \to (\widetilde{\mathsf{sk}}_f, \widetilde{\mathsf{vk}}_f, \widetilde{\mathsf{ek}}_f)$: *the key generation algorithm takes as input a security parameter λ and the description of a function f, and outputs a secret key $\widetilde{\mathsf{sk}}_f$ that will be used for input delegation, a corresponding verification key $\widetilde{\mathsf{vk}}_f$, and an evaluation key $\widetilde{\mathsf{ek}}_f$, which will be used for the evaluation of f.*
- $\mathsf{VC.ProbGen}(\widetilde{\mathsf{sk}}_f, \mathsf{m}) \to (\widetilde{\sigma}_\mathsf{m}, \widetilde{\rho}_\mathsf{m})$: *the problem generation algorithm uses the secret key $\widetilde{\mathsf{sk}}_f$ to encode the function input m as an encoded value $\widetilde{\sigma}_\mathsf{m}$ and a corresponding decoding value $\widetilde{\rho}_\mathsf{m}$.*
- $\mathsf{VC.Compute}(\widetilde{\mathsf{ek}}_f, \widetilde{\sigma}_\mathsf{m}) \to \widetilde{\sigma}_y$: *the computing algorithm takes as input the evaluation key $\widetilde{\mathsf{ek}}_f$ and the encoded input $\widetilde{\sigma}_\mathsf{m}$ and outputs $\widetilde{\sigma}_y$, an encoded version of the function's output $y = f(\mathsf{m})$.*
- $\mathsf{VC.Verify}(\widetilde{\mathsf{vk}}_f, \widetilde{\rho}_\mathsf{m}, \widetilde{\sigma}_y) \to y$ or $\perp$: *the verification algorithm takes as input the verification key $\widetilde{\mathsf{vk}}_f$, the decoding value $\widetilde{\rho}_\mathsf{m}$ and the encoded output $\widetilde{\sigma}_y$. The algorithm outputs y if and only if $y = f(\mathsf{m})$ is correctly computed. Otherwise $\perp$ is the output.*

A publicly verifiable computation scheme is a VC scheme with an additional property that the verification key $\widetilde{\mathsf{vk}}_f$ is published publicly such that anyone can check the correctness of a performed computation.

Remark 1. *The original VC [8] is with* "secret-key" *nature. In the earlier definition,* KeyGen *produces a secret key that was used as an input to* ProbGen *and, in turn,* ProbGen *produces a* secret *verification value needed for* Verify*. Later, Parno* et al. *[14] introduced the* "public-key" *VC definition which has both the* public delegation ***and*** public verification *properties. The delegation being public or private depends on whether the evaluation key $\widetilde{\mathsf{sk}}$ is published or kept secret. In our case, we consider the scenario where the* Public Verifiable Computation (PVC) *scheme is publicly verifiable but privately delegatable,* i.e. *the evaluation key $\widetilde{\mathsf{ek}}_f$ is secret while the verification key $\widetilde{\mathsf{vk}}_f$ is public. In the paper, we abuse terminology and refer to a PVC scheme when discussing about a* Verifiable Computation (VC) *scheme.*

Correctness: a verifiable computation scheme VC is **correct** for a class of functions $\mathcal{F}$ if for any $f \in \mathcal{F}$, for any tuple of keys $(\widetilde{\mathsf{sk}}_f, \widetilde{\mathsf{vk}}_f, \widetilde{\mathsf{ek}}_f) \leftarrow \mathsf{VC.KeyGen}(\lambda, f)$, for any $\mathsf{m} \in \mathcal{D}_f$, for any $(\widetilde{\sigma}_\mathsf{m}, \widetilde{\rho}_\mathsf{m}) \leftarrow \mathsf{VC.ProbGen}(\widetilde{\mathsf{sk}}_f, \mathsf{m})$ and any computed $\widetilde{\sigma}_y$ obtained from $\mathsf{VC.Compute}(\widetilde{\mathsf{ek}}_f, \widetilde{\sigma}_\mathsf{m})$, it holds that $\mathsf{VC.Verify}(\widetilde{\mathsf{vk}}_f, \widetilde{\rho}_\mathsf{m}, \widetilde{\sigma}_y) = y = f(\mathsf{m})$.

Security: a VC scheme is *secure w.r.t.* a static attacker if the probability of any PPT algorithm $\mathcal{A}$ in the VC static security experiment $\mathsf{Exp}^{\mathsf{VC.StaticVerify}}_{\mathsf{VC}}(\mathcal{A})$ of Figure 4, to output 1 is negligible. Namely,

$$\mathsf{Adv}^{\mathsf{VC.StaticVerify}}_{\mathcal{A},\mathsf{VC}}(\lambda) = \Pr\left(\mathsf{Exp}^{\mathsf{VC.StaticVerify}}_{\mathsf{VC}}(\mathcal{A}) = 1\right) \leq \mathsf{negl}(\lambda)$$

Privacy [9]: a VC scheme is said to be *private w.r.t.* a static attacker if the advantage of any PPT algorithm $\mathcal{A}$ winning in the VC privacy experiment $\mathsf{Exp}^{\mathsf{VC.Priv}}_{\mathsf{VC}}(\mathcal{A})$ of Figure 4 is negligible. Namely,

$$\mathsf{Adv}^{\mathsf{VC.Priv}}_{\mathcal{A},\mathsf{VC}}(\lambda) = \left|\Pr\left(\mathsf{Exp}^{\mathsf{VC.Priv}}_{\mathsf{VC}}(\mathcal{A}) = 1\right) - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$$

$\mathsf{Exp}^{\mathsf{VC.StaticVerify}}_{\mathsf{VC}}(\mathcal{A})$

$f \leftarrow \mathcal{A}\,(1^n)$

$(\widetilde{\mathsf{sk}}_f, \widetilde{\mathsf{vk}}_f, \widetilde{\mathsf{ek}}_f) \leftarrow \mathsf{VC.KeyGen}\,(\lambda, f)$

$(\widetilde{\sigma}_0, \widetilde{\rho}_0) = (\emptyset, \emptyset)$

for $i \in \{1, \ldots, t = \mathsf{poly}(\lambda)\}$ **do**

$\mathsf{m}_i \leftarrow \mathcal{A}\left(\begin{smallmatrix}\widetilde{\mathsf{ek}}_f, \widetilde{\rho}_1, \ldots, \widetilde{\rho}_{i-1} \\ \widetilde{\mathsf{vk}}_f, \widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_{i-1}\end{smallmatrix}\right)$

$(\widetilde{\sigma}_i, \widetilde{\rho}_i) \leftarrow \mathsf{VC.ProbGen}(\widetilde{\mathsf{sk}}_f, \mathsf{m}_i)$

$\mathsf{m}^\star \leftarrow \mathcal{A}\left(\begin{smallmatrix}\widetilde{\mathsf{ek}}_f, \widetilde{\rho}_1, \ldots, \widetilde{\rho}_t \\ \widetilde{\mathsf{vk}}_f, \widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_t\end{smallmatrix}\right)$

$(\widetilde{\sigma}, \widetilde{\rho}) \leftarrow \mathsf{VC.ProbGen}(\widetilde{\mathsf{sk}}_f, \mathsf{m}^\star)$

$\widetilde{\sigma}^\star \leftarrow \mathcal{A}\left(\begin{smallmatrix}\widetilde{\mathsf{ek}}_f, \widetilde{\rho}_1, \ldots, \widetilde{\rho}_t, \widetilde{\rho} \\ \widetilde{\mathsf{vk}}_f, \widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_t, \widetilde{\sigma}\end{smallmatrix}\right)$

$y^\star \leftarrow \mathsf{VC.Verify}(\widetilde{\mathsf{vk}}_f, \widetilde{\rho}, \widetilde{\sigma}^\star)$

if $(y^\star \neq \perp) \wedge (y^\star \neq f(\mathsf{m}^\star))$

then return 1

else return 0

$\mathsf{Exp}^{\mathsf{VC.Priv}}_{\mathsf{VC}}(\mathcal{A})$

$(f_0, f_1, \mathsf{m}_0, \mathsf{m}_1) \leftarrow \mathcal{A}\,(1^n)$

if $f_0(\mathsf{m}_0) \neq f_1(\mathsf{m}_1)$ **then**

return $\perp$

$b \leftarrow\$ \{0, 1\}$

$(\widetilde{\mathsf{sk}}_{f_b}, \widetilde{\mathsf{vk}}_{f_b}, \widetilde{\mathsf{ek}}_{f_b}) \leftarrow \mathsf{VC.KeyGen}\,(\lambda, f_b)$

$(\widetilde{\sigma}_b, \widetilde{\rho}_b) \leftarrow \mathsf{VC.ProbGen}(\widetilde{\mathsf{sk}}_{f_b}, \mathsf{m}_b)$

$\widetilde{\sigma}_{y_b} \leftarrow \mathsf{VC.Compute}(\widetilde{\mathsf{ek}}_{f_b}, \widetilde{\sigma}_b)$

$b^\star \leftarrow \mathcal{A}(\widetilde{\mathsf{vk}}_{f_b}, \widetilde{\sigma}_{y_b}, \widetilde{\rho}_b, f_0, f_1, \mathsf{m}_0, \mathsf{m}_1)$

if $b^\star = b$ **then return** 1

else return 0

Fig. 4: VC static security and privacy experiments.

2.6 Fiore-Gennaro's PVC Scheme

Fiore and Gennaro [8] propose a publicly VC scheme for the function family $\mathcal{F}$ containing all multivariate polynomials $f(x_1, \ldots, x_m)$ with coefficients in $\mathbb{Z}_p$ for some prime p, m variables and degree at most d in each variable. Let $h : \mathbb{Z}_p^m \to \mathbb{Z}_p^l$ which expands the input $\vec{x}$ to the vector $(h_1(\vec{x}), \ldots, h_l(\vec{x}))$ of all the monomials as follows: for all $j \in \{1, \ldots, l\}$ where $l = (d+1)^m$, write $j = (i_1, \ldots, i_m)$ with $i_k \in \{0, \ldots, d\}$, then $h_j(\vec{x}) = (x_1^{i_1} \cdots x_m^{i_m})$. Thus, by using this notation, it is possible to write the polynomial as $f(\vec{x}) = \langle \vec{f}, h(\vec{x}) \rangle = \sum_{j=1}^{l} f_j \cdot h_j(\vec{x})$ where the f_j's are its coefficients and $f_j \in \mathbb{Z}_p$. The construction works over the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of the same prime order p, equipped with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let us define $\mathsf{Poly}(\vec{R}, \vec{x}) = \prod_{j=1}^{l} R_j^{h_j(\vec{x})}$ where $\vec{R}$ is a random l-dimensional vector of $\mathbb{G}_1^l$.

Let $\mathsf{CF} = (\mathsf{CF.KeyGen}, \mathsf{CF.H}, \mathsf{CF.Eval})$ be a CFE PRF defined in Section 2.2. Fiore-Gennaro's public verifiable computation scheme [8] VC is constructed as the follows:

- $\mathsf{VC.KeyGen}(\lambda, f) \to (\widetilde{\mathsf{sk}}_f, \widetilde{\mathsf{vk}}_f, \widetilde{\mathsf{ek}}_f)$: Generate the description of a bilinear group $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ in the security parameter λ, a key of a PRF $K \leftarrow \mathsf{CF.KeyGen}(\lambda, \lceil \log d \rceil, m)$ with range in $\mathbb{G}_1$. Randomly sample $\alpha \leftarrow\$ \mathbb{Z}_p$ and, for all the indexes $i \in \{1, \ldots, l\}$, compute $W_i = g_1^{\alpha \cdot f_i} \mathsf{CF.H}_K(i)$ and define W as $(W_1, \ldots, W_l) \in \mathbb{G}_1^l$. Output the key tuple $(\widetilde{\mathsf{sk}}_f, \widetilde{\mathsf{vk}}_f, \widetilde{\mathsf{ek}}_f)$ as the values $\left(K, e\,(g_1, g_2)^\alpha, (f, W)\right)$.
- $\mathsf{VC.ProbGen}(\widetilde{\mathsf{sk}}_f, \vec{\mathsf{m}}) \to (\widetilde{\sigma}_\mathsf{m}, \widetilde{\rho}_\mathsf{m})$: Output the tuple $(\widetilde{\sigma}_\mathsf{m}, \widetilde{\rho}_\mathsf{m})$ where $\widetilde{\sigma}_\mathsf{m} = \vec{\mathsf{m}}$ and $\widetilde{\rho}_\mathsf{m} = e\left(\mathsf{CF.Eval}_{\mathsf{Poly}}(K, h(\vec{\mathsf{m}})), g_2\right)$.
- $\mathsf{VC.Compute}(\widetilde{\mathsf{ek}}_f, \widetilde{\sigma}_\mathsf{m}) \to \widetilde{\sigma}_y$: Compute y by evaluating $f(\vec{\mathsf{m}}) = \sum_{i=1}^{l} f_i h_i(\vec{\mathsf{m}})$ and $V = \prod_{i=1}^{l} W_i^{h_i(\vec{\mathsf{m}})}$. Output $\widetilde{\sigma}_y = (y, V)$.
- $\mathsf{VC.Verify}(\widetilde{\mathsf{vk}}_f, \widetilde{\rho}_\mathsf{m}, \widetilde{\sigma}_y) \to \{y, \perp\}$: output y if it holds that $e\,(V, g_2) \stackrel{?}{=} \left(\widetilde{\mathsf{vk}}_f\right)^y \cdot \widetilde{\rho}_\mathsf{m}$. Otherwise output $\perp$.

Fiore and Gennaro [8] proved that the construction is secure if the co-CDH assumption holds and $\mathsf{CF.PRF}$ is a close form efficient PRF. In Lemma 1, we prove that Fiore-Gennaro PVC scheme satisfies *privacy* as defined in the experiment depicted in Figure 4.

3 Construction Blocks: Variated Schemes

In this section, we provide our variations of the Boneh-Lynn-Shacham signature scheme [5] and Fiore-Gennaro publicly verifiable computation scheme [8].

In a nutshell, the variations add to the schemes a *"setup algorithm"* that outputs a master key-pair used in the original key-generation algorithm and in the final verification algorithm while the accordingly modified security games reduce to the ones of the original schemes. The final purpose of these modifications is to later allowing the instantiation of both the two schemes with a *single* common master key-pair in a stronger security setting, where the master secret-key is kept secure as in the act of *"merging"* the schemes into a single one. Intuitively, with the shared schemes' master public-key, the final verification algorithm will compute the two schemes' verification algorithms independently **and** will verify that the schemes are indeed *"merged"* into a single one.

3.1 A variation of the BLS signature

We introduce, in the BLS signature scheme, a Setup algorithm that outputs a master key-pair $(\mathsf{MPK}, \mathsf{MSK})$ used in the KeyGen algorithm to produce a local signing key in order to generate a signature for a message together with a local verification key. The Verify algorithm will take both the master public key and the local verification key to check the validity of a message-signature pair. We provide the unforgeability game for our BLS variation in Figure 5 and prove the unforgeability of it in the random oracle model.

Definition 4 (BLS Variation)**.** *Let $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map in the security parameter λ. Let $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$ be a full-domain hash function and $F : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$ a PRF. Let the additional information $\alpha \in \mathbb{Z}_p$ be a field element known just to the signer. Our variation $\overline{\mathsf{BLS}}$ scheme is defined as the algorithms:*

- $\overline{\mathsf{BLS}}.\mathsf{Setup}(\lambda) \to (\mathsf{MPK}, \mathsf{MSK})$*: sample $\beta \leftarrow\$ \mathbb{Z}_p$, set $\mathsf{MSK} = \beta$. Compute $\mathsf{MPK} = e\,(g_1, g_2)^\beta$ and output $(\mathsf{MPK}, \mathsf{MSK}) \in \mathbb{G}_T \times \mathbb{Z}_p$.*
- $\overline{\mathsf{BLS}}.\mathsf{KeyGen}(\mathsf{MSK}, \alpha) \to (\mathsf{PK}, \mathsf{SK})$*: given $\mathsf{MSK} \in \mathbb{Z}_p$ and $\alpha \in \mathbb{Z}_p$, sample $k \leftarrow\$ \mathbb{Z}_p$, $r \in \mathbb{Z}_p$ and compute secret key as $\mathsf{SK} = (\mathsf{SK}_1, \mathsf{SK}_2) = (g_1^{\mathsf{MSK}+\alpha+r}, k)$ and the public key as $\mathsf{PK} = (\mathsf{PK}_1, \mathsf{PK}_2) = (e\,(g_1, g_2)^{\alpha+r}, g_2^{\mathsf{SK}_2})$.*

- $\overline{\mathit{BLS}}.\mathit{Sign}(\mathsf{SK}, m) \to \ddot{\sigma}$: *given a secret key* $\mathsf{SK} = (\mathsf{SK}_1, \mathsf{SK}_2)$ *and a message* $\mathsf{m} \in \mathcal{M}$, *compute and output the signature* $\ddot{\sigma} = \mathsf{SK}_1 \cdot \mathsf{H}(m)^{\mathsf{SK}_2}$.
- $\overline{\mathit{BLS}}.\mathit{Verify}(\mathsf{MPK}, \mathsf{PK}, m, \ddot{\sigma}) \to \{0,1\}$: *given a public key* $\mathsf{PK} = (\mathsf{PK}_1, \mathsf{PK}_2)$, *a message* m, *a signature* $\ddot{\sigma}$ *and a environmental public key* MPK, *verify and output the result of the check* $e(\ddot{\sigma}, g_2) \stackrel{?}{=} \mathsf{MPK} \cdot \mathsf{PK}_1 \cdot e(\mathsf{H}(m), \mathsf{PK}_2)$.

$\mathsf{Exp}^{\mathsf{BLS.UNF}}_{\mathsf{BLS}}(\mathcal{A})$

$(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{BLS.KeyGen}(\lambda)$

$(\mathsf{m}^\star, \ddot{\sigma}^\star) \leftarrow \mathcal{A}^{\mathsf{O}_{\mathsf{BLS.Sign}(\mathsf{SK},\cdot)}, \mathsf{O}_{\mathsf{H}}(\cdot)}(\mathsf{PK})$

if $(\mathsf{m}^\star, \ddot{\sigma}^\star) \in \mathcal{L}_\Delta$ **then return** $\perp$

return $\mathsf{BLS.Verify}(\mathsf{PK}, \mathsf{m}^\star, \ddot{\sigma}^\star)$

$\mathsf{Exp}^{\overline{\mathsf{BLS}}.\mathsf{UNF}}_{\overline{\mathsf{BLS}}}(\mathcal{A})$

$(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \overline{\mathsf{BLS}}.\mathsf{Setup}(\lambda)$

$\alpha^\star \leftarrow \mathcal{A}(\mathsf{MPK}); \quad \mathcal{L}_\Delta = \emptyset$

$(\mathsf{PK}, \mathsf{SK}) \leftarrow \overline{\mathsf{BLS}}.\mathsf{KeyGen}(\lambda, \mathsf{MSK}, \alpha^\star)$

$(\mathsf{m}^\star, \ddot{\sigma}^\star) \leftarrow \mathcal{A}^{\mathsf{O}_{\overline{\mathsf{BLS}}.\mathsf{Sign}}(\mathsf{SK},\cdot), \mathsf{O}_{\mathsf{H}}(\cdot)}(\mathsf{MPK}, \mathsf{PK})$

if $(\mathsf{m}^\star, \ddot{\sigma}^\star) \in \mathcal{L}_\Delta$ **then return** $\perp$

return $\overline{\mathsf{BLS}}.\mathsf{Verify}(\mathsf{MPK}, \mathsf{PK}, \mathsf{m}^\star, \ddot{\sigma}^\star)$

Fig. 5: BLS and $\overline{\mathsf{BLS}}$ unforgeability experiments.

We present in Figure 5 a modified unforgeability experiment for the $\overline{\mathsf{BLS}}$ scheme which, differently from the BLS standard unforgeability experiment, must consider the generation of the master key pair and the value $\alpha^\star$. We prove that, despite the modification, unforgeability is preserved.

Proposition 1. *If the advantage for all PPT adversaries* $\mathcal{B}$ *for the experiment* $\mathit{Exp}^{\mathit{BLS.UNF}}_{\mathit{BLS}}(\mathcal{B})$ *is negligible, then all the PPT adversaries* $\mathcal{A}$ *for the experiment* $\mathit{Exp}^{\overline{\mathit{BLS}}.\mathit{UNF}}_{\overline{\mathit{BLS}}}(\mathcal{A})$ *have a negligible advantage. Formally:*

$$\mathsf{Adv}^{\overline{\mathit{BLS}}.\mathit{UNF}}_{\mathcal{A},\overline{\mathit{BLS}}}(\lambda) \leq \mathsf{Adv}^{\mathit{BLS.UNF}}_{\mathcal{B},\mathit{BLS}}(\lambda) \leq \mathsf{negl}(\lambda)$$

Proof: let us assume that there exists a PPT adversary $\mathcal{A}$ for the experiment $\mathsf{Exp}^{\overline{\mathsf{BLS}}.\mathsf{UNF}}_{\overline{\mathsf{BLS}}}(\mathcal{A})$ with non-negligible advantage Δ. The oracles $\mathsf{O}_{\mathsf{BLS.Sign(SK)}}(\mathsf{m})$ and $\mathsf{O}_{\overline{\mathsf{BLS}}.\mathsf{Sign(SK)}}(\mathsf{m})$ is to respond with the signatures on the messages m submitted to each challenger and then keep a track of the message-signature pair in its queried set $\mathcal{L}_\Delta$. Now we construct an adversary $\mathcal{R}$, running $\mathcal{A}$ as a subroutine, which attacks the underlying BLS scheme. Receiving from $\underline{\mathsf{BLS}}$ challenger the public key $\mathsf{PK}_\star$, $\mathcal{R}$ sets it to be PK_2. $\mathcal{R}$ runs $\overline{\mathsf{BLS}}.\mathsf{Setup}(\lambda) \to (\mathsf{MPK}, \mathsf{MSK})$. It then outputs MPK to $\mathcal{A}$. $\mathcal{A}$ will reply with ξ and α. $\mathcal{R}$ fixes $\mathsf{SK}_1 = g_1^{\mathsf{MSK}+\alpha+r}$ and computes $\mathsf{PK}_1 = e\left(g_1^{\mathsf{MSK}+\alpha+r}, g_2\right)$ and outputs $\mathsf{PK} = (\mathsf{PK}_1, \mathsf{PK}_2)$ to $\mathcal{A}$. After the key generation phase, for every signing query $\mathsf{O}_{\overline{\mathsf{BLS}}.\mathsf{Sign}}(\mathsf{m})$ from $\mathcal{A}$, the reduction $\mathcal{R}$ queries $\mathcal{B}$'s oracle with $\mathsf{O}_{\mathsf{BLS.Sign}}(\mathsf{m})$ and obtains $\ddot{\sigma}_\star$. For any hash query $\mathsf{O}_{\mathsf{H}}(\mathsf{m})$ from $\mathcal{A}$, $\mathcal{R}$ queries $\mathcal{B}$'s hash oracle with $\mathsf{O}_{\mathsf{H}}(\mathsf{m})$ and obtains $\mathsf{H}(\mathsf{m})$. $\mathcal{R}$ computes $\ddot{\sigma} = \mathsf{SK}_1 \cdot \ddot{\sigma}_\star$ and returns it to $\mathcal{A}$. When $\mathcal{A}$ outputs the forgery $(\mathsf{m}^\star, \ddot{\sigma}^\star)$, the reduction $\mathcal{R}$ outputs $(\mathsf{m}^\star, \ddot{\sigma}^\star \cdot g_1^{-\alpha-\mathsf{MSK}-r})$.

It is direct to check that $\mathcal{R}$ output is a correct forgery for the BLS signature scheme since:

$$\mathsf{BLS.Verify}(\mathsf{PK}_\star, \mathsf{m}^\star, \ddot{\sigma}^\star \cdot g_1^{-\alpha-\mathsf{MSK}-r}) \Leftrightarrow$$

$$e\left(\ddot{\sigma}^\star \cdot g_1^{-\alpha-\mathsf{MSK}-r}, g_2\right) \stackrel{?}{=} e\left(\mathsf{H}(\mathsf{m}^\star), \mathsf{PK}_\star\right) \Leftrightarrow$$

$$\Leftrightarrow e\left(\ddot{\sigma}^\star, g_1\right) \stackrel{?}{=} e(g_1, g_2)^{\alpha+\mathsf{MSK}+r} \cdot e\left(\mathsf{H}(\mathsf{m}^\star), \mathsf{PK}_\star\right)$$

$$\Leftrightarrow e\left(\ddot{\sigma}^\star, g_1\right) \stackrel{?}{=} \mathsf{MPK} \cdot \mathsf{PK}_1 \cdot e\left(\mathsf{H}(\mathsf{m}^\star), \mathsf{PK}_2\right)$$

$$\Leftrightarrow \overline{\mathsf{BLS}}.\mathsf{Verify}(\mathsf{MPK}, \mathsf{PK}, \mathsf{m}^\star, \ddot{\sigma}^\star)$$

therefore $\Delta = \mathsf{Adv}^{\overline{\mathsf{BLS}}.\mathsf{UNF}}_{\mathcal{A},\overline{\mathsf{BLS}}}(\lambda) \leq \mathsf{Adv}^{\mathsf{BLS.UNF}}_{\mathcal{B},\mathsf{BLS}}(\lambda)$ which is a contradiction. □

3.2 A variation of Fiore-Gennaro's PVC

In our PVC variation, we introduce a *master key-pair* $(\widetilde{\mathsf{msk}}, \widetilde{\mathsf{mpk}})$ that is generated in the Setup phase and set as $(\beta, e(g_1, g_2)^\beta)$, which adds additional randomness to the evaluation key of function f such that W_i in Fiore-Gennaro's PVC is rerandomized to $W_i \cdot g_1^{\beta \cdot f_i}$. By forcing the master secret-key to be zero, *i.e.* $\beta = 0$, we obtain the original Fiore-Gennaro's scheme.

Definition 5 (Fiore-Gennaro PVC Variation)**.** *Let* pp *be the description of a bilinear group* $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ *in the security parameter* λ. *Our publicly verifiable computation scheme* $\overline{\mathsf{VC}}$ *is defined by the following algorithms:*

- $\overline{\mathsf{VC}}.\mathsf{Setup}(\lambda) \to (\widetilde{\mathsf{msk}}, \widetilde{\mathsf{mpk}})$: *the setup algorithm randomly sample* $\beta \leftarrow_\$ \mathbb{Z}_p$ *and outputs* $\left(\widetilde{\mathsf{msk}}, \widetilde{\mathsf{mpk}}\right) = \left(\beta, e(g_1, g_2)^\beta\right)$.
- $\overline{\mathsf{VC}}.\mathsf{KeyGen}(\lambda, \widetilde{\mathsf{msk}}, f) \to (\widetilde{\mathsf{sk}}_f, \widetilde{\mathsf{vk}}_f, \widetilde{\mathsf{ek}}_f)$: *let* $\widetilde{\mathsf{msk}} = \beta$. *The algorithm samples* $\alpha \leftarrow_\$ \mathbb{Z}_p$ *and generates a PRF key* $K \leftarrow \mathsf{CF.KeyGen}(\lambda, \lceil \log d \rceil, m)$ *with range in* $\mathbb{G}_1$. *For all* $i \in \{1, \ldots, l\}$, *it computes* $W_i = g_1^{(\alpha+\beta) \cdot f_i} \mathsf{CF.H}_K(i)$ *and let* W *be defined as* $(W_1, \ldots, W_l) \in \mathbb{G}_1^l$. *It outputs* $\left(\widetilde{\mathsf{sk}}_f, \widetilde{\mathsf{vk}}_f, \widetilde{\mathsf{ek}}_f\right)$ *as* $\left((\alpha, g_2^\alpha, K), e(g_1, g_2)^\alpha, (f, W)\right)$.
- $\overline{\mathsf{VC}}.\mathsf{ProbGen}(\widetilde{\mathsf{sk}}_f, \vec{m}) \to (\widetilde{\sigma}_m, \widetilde{\rho}_m)$: *Output the tuple* $\left(\widetilde{\sigma}_m, \widetilde{\rho}_m\right)$ *where* $\widetilde{\sigma}_m = \vec{m}$ *and* $\widetilde{\rho}_m = e\left(\mathsf{CF.Eval}_{\mathsf{Poly}}(K, h(\vec{m})), g_2^\alpha\right)$.
- $\overline{\mathsf{VC}}.\mathsf{Compute}(\widetilde{\mathsf{ek}}_f, \widetilde{\sigma}_m) \to \widetilde{\sigma}_y$: *Compute* y *by evaluating* $f(\vec{m}) = \sum_{i=1}^l f_i h_i(\vec{m})$ *and* $V = \prod_{i=1}^l W_i^{h_i(\vec{m})}$. *Output* $\widetilde{\sigma}_y = (y, V)$.
- $\overline{\mathsf{VC}}.\mathsf{Verify}(\widetilde{\mathsf{mpk}}, \widetilde{\mathsf{vk}}_f, \widetilde{\rho}_m, \widetilde{\sigma}_y) \to \{y, \perp\}$: *the algorithm checks if it holds that* $e(V, g_2) \stackrel{?}{=} \left(\widetilde{\mathsf{vk}}_f \cdot \widetilde{\mathsf{mpk}}\right)^y \cdot \widetilde{\rho}_m$. *If it is true, then it outputs* y. *Otherwise it outputs* $\perp$.

Remark 2. *It seems redundant to include* α *in* $\widetilde{\mathsf{sk}}_f$, *since the component of* (g_2^α, K) *suffices to obtain* $(\widetilde{\sigma}_m, \widetilde{\rho}_m)$. *However, looking ahead, the component* α *of* $\widetilde{\mathsf{sk}}_f$ *plays the role of building a bridge between* $\overline{\mathsf{VC}}$*and* $\overline{\mathsf{BLS}}$*in order to achieve an SFS.*

We describe the security and privacy experiments in Fig. 6.

Proposition 2. *If all PPT adversaries* $\mathcal{B}$ *for the experiment* $\mathit{Exp}^{\mathit{VC.StaticVerify}}_{\mathit{VC}}(\mathcal{B})$ *have a negligible advantage, then all the PPT adversaries* $\mathcal{A}$ *for the experiment* $\mathit{Exp}^{\overline{\mathit{VC}}.\mathit{StaticVerify}}_{\overline{\mathit{VC}}}(\mathcal{A})$ *have a negligible advantage. Formally:*

$$\mathsf{Adv}^{\overline{\mathit{VC}}.\mathit{StaticVerify}}_{\mathcal{A},\overline{\mathit{VC}}}(\lambda) \leq \mathsf{Adv}^{\mathit{VC.StaticVerify}}_{\mathcal{B},\mathit{VC}}(\lambda) \leq \mathsf{negl}(\lambda)$$

and, mutatis mutandis*, it holds:*

$$\mathsf{Adv}^{\overline{\mathit{VC}}.\mathit{Priv}}_{\mathcal{A},\overline{\mathit{VC}}}(\lambda) \leq \mathsf{Adv}^{\mathit{VC.Priv}}_{\mathcal{B},\mathit{VC}}(\lambda)$$

Proof: let us assume by contradiction that there exists a PPT adversary $\mathcal{A}$ for the experiment $\mathsf{Exp}^{\overline{\mathsf{VC}}.\mathsf{StaticVerify}}_{\overline{\mathsf{VC}}}(\mathcal{A})$ with non-negligible advantage Δ. We build an adversary $\mathcal{R}$, running $\mathcal{A}$ as a subroutine, which attacks the security of the underlying VC scheme. $\mathcal{R}$ runs $\overline{\mathsf{VC}}.\mathsf{Setup}(\lambda) \to (\widetilde{\mathsf{mpk}}, \widetilde{\mathsf{msk}})$ and then outputs $\widetilde{\mathsf{mpk}}$ to $\mathcal{A}$ that will

$\mathsf{Exp}^{\overline{\mathsf{VC}}.\mathsf{StaticVerify}}_{\overline{\mathsf{VC}}}(\mathcal{A})$

$(\widetilde{\mathsf{mpk}}, \widetilde{\mathsf{msk}}) \leftarrow \overline{\mathsf{VC}}.\mathsf{Setup}(\lambda)$

$f \leftarrow \mathcal{A}\left(\boxed{\widetilde{\mathsf{mpk}}}\right)$

$(\widetilde{\mathsf{sk}}_f, \widetilde{\mathsf{vk}}_f, \widetilde{\mathsf{ek}}_f) \leftarrow \overline{\mathsf{VC}}.\mathsf{KeyGen}\left(\begin{array}{c}\lambda, f \\ \boxed{\widetilde{\mathsf{msk}}}\end{array}\right)$

for $i \in \{1, \ldots, t = \mathsf{poly}(\lambda)\}$ **do**

$\mathsf{m}_i \leftarrow \mathcal{A}\left(\begin{array}{c}\widetilde{\mathsf{ek}}_f, \widetilde{\rho}_1, \ldots, \widetilde{\rho}_{i-1} \\ \widetilde{\mathsf{vk}}_f, \widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_{i-1}\end{array}, \boxed{\widetilde{\mathsf{mpk}}}\right)$

$(\widetilde{\sigma}_i, \widetilde{\rho}_i) \leftarrow \overline{\mathsf{VC}}.\mathsf{ProbGen}(\widetilde{\mathsf{sk}}_f, \mathsf{m}_i)$

$\mathsf{m}^\star \leftarrow \mathcal{A}\left(\begin{array}{c}\widetilde{\mathsf{ek}}_f, \widetilde{\rho}_1, \ldots, \widetilde{\rho}_t \\ \widetilde{\mathsf{vk}}_f, \widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_t\end{array}, \boxed{\widetilde{\mathsf{mpk}}}\right)$

$(\widetilde{\sigma}, \widetilde{\rho}) \leftarrow \overline{\mathsf{VC}}.\mathsf{ProbGen}(\widetilde{\mathsf{sk}}_f, \mathsf{m}^\star)$

$\widetilde{\sigma}^\star \leftarrow \mathcal{A}\left(\begin{array}{c}\widetilde{\mathsf{ek}}_f, \widetilde{\rho}_1, \ldots, \widetilde{\rho}_t, \widetilde{\rho} \\ \widetilde{\mathsf{vk}}_f, \widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_t, \widetilde{\sigma}\end{array}, \boxed{\widetilde{\mathsf{mpk}}}\right)$

$y^\star \leftarrow \overline{\mathsf{VC}}.\mathsf{Verify}(\widetilde{\mathsf{vk}}_f, \widetilde{\rho}, \widetilde{\sigma}^\star)$

if $(y^\star \neq \perp) \wedge (y^\star \neq f(\mathsf{m}^\star))$

then return 1

else return 0

$\mathsf{Exp}^{\overline{\mathsf{VC}}.\mathsf{Priv}}_{\overline{\mathsf{VC}}}(\mathcal{A})$

$(\widetilde{\mathsf{mpk}}, \widetilde{\mathsf{msk}}) \leftarrow \overline{\mathsf{VC}}.\mathsf{Setup}(\lambda)$

$b \leftarrow\$ \{0, 1\}$

$(f_0, f_1, \mathsf{m}_0, \mathsf{m}_1) \leftarrow \mathcal{A}\left(\boxed{\widetilde{\mathsf{mpk}}}\right)$

if $f_0(\mathsf{m}_0) \neq f_1(\mathsf{m}_1)$ **then**

return $\perp$

$(\widetilde{\mathsf{sk}}_{f_b}, \widetilde{\mathsf{vk}}_{f_b}, \widetilde{\mathsf{ek}}_{f_b}) \leftarrow \mathsf{VC.KeyGen}\left(\begin{array}{c}\lambda, f_b \\ \boxed{\widetilde{\mathsf{msk}}}\end{array}\right)$

$(\widetilde{\sigma}_b, \widetilde{\rho}_b) \leftarrow \mathsf{VC.ProbGen}(\widetilde{\mathsf{sk}}_{f_b}, \mathsf{m}_b)$

$\widetilde{\sigma}_{y_b} \leftarrow \mathsf{VC.Compute}(\widetilde{\mathsf{ek}}_{f_b}, \widetilde{\sigma}_b)$

$b^\star \leftarrow \mathcal{A}(\boxed{\widetilde{\mathsf{mpk}}}, \widetilde{\mathsf{vk}}_{f_b}, \widetilde{\sigma}_{y_b}, \widetilde{\rho}_b, f_0, f_1, \mathsf{m}_0, \mathsf{m}_1)$

if $b^\star = b$ **then return** 1

else return 0

Fig. 6: The static security and privacy experiments for $\overline{\mathsf{VC}}$ scheme. In $\boxed{\text{box}}$ are high-lighted the variations introduced in the $\overline{\mathsf{VC}}$ experiments in comparison to the original Fiore-Gennaro VC scheme.

reply with the challenging function f. The reduction $\mathcal{R}$ just forwards it to the challenger of VC scheme and obtains $(\widetilde{\mathsf{vk}}_f, \widetilde{\mathsf{ek}}_f)$ where $\widetilde{\mathsf{ek}}_f = (f, W_\star)$. $\mathcal{R}$ modifies $W_\star$ into W by computing, for all $i \in \{1, \ldots, l\}$, the new values $W_i = W_{i\star} \cdot g_1^{\widetilde{\mathsf{msk}} \cdot f_i}$. It then returns $(\widetilde{\mathsf{vk}}_f, (f, W))$ to $\mathcal{A}$. All the ProbGen queries from $\mathcal{A}$ are just forwarded to the challenged of VC scheme and are responded with the same response from VC challenger. When the adversary $\mathcal{A}$ outputs the forgery $(i^\star, \widetilde{\sigma}^\star)$ where $\widetilde{\sigma}^\star = (y^\star, V^\star)$, the reduction $\mathcal{R}$ and outputs $(i^\star, (y^\star, V^\star \cdot g_1^{-\widetilde{\mathsf{msk}} \cdot y}))$. It is straightforward to check that $\mathcal{R}$ output is a correct tamper for the VC scheme since:

$$\mathsf{VC.Verify}\left(\widetilde{\mathsf{vk}}_f, \widetilde{\rho}_{i^\star}, \left(y^\star, V^\star \cdot g_1^{-\widetilde{\mathsf{msk}} \cdot y}\right)\right) \Leftrightarrow$$

$$e\left(V^\star \cdot g_1^{-\widetilde{\mathsf{msk}} \cdot y}, g_2\right) \stackrel{?}{=} \widetilde{\mathsf{vk}}_f^{\,y} \cdot \widetilde{\rho}_{i^\star} \Leftrightarrow$$

$$e\left(V^\star, g_2\right) e\left(g_1, g_2\right)^{-\widetilde{\mathsf{msk}} \cdot y} \stackrel{?}{=} \widetilde{\mathsf{vk}}_f^{\,y} \cdot \widetilde{\rho}_{i^\star} \Leftrightarrow$$

$$\Leftrightarrow e\left(V^\star, g_2\right) \stackrel{?}{=} e\left(g_1, g_2\right)^{\widetilde{\mathsf{msk}} \cdot y} \cdot \widetilde{\mathsf{vk}}_f^{\,y} \cdot \widetilde{\rho}_{i^\star}$$

$$\Leftrightarrow e\left(V^\star, g_2\right) \stackrel{?}{=} \left(\widetilde{\mathsf{mpk}} \cdot \widetilde{\mathsf{vk}}_f\right)^y \cdot \widetilde{\rho}_{i^\star}$$

$$\Leftrightarrow \overline{\mathsf{VC}}.\mathsf{Verify}(\widetilde{\mathsf{mpk}}, \widetilde{\mathsf{vk}}_f, \widetilde{\rho}_i^\star, \widetilde{\sigma}^\star)$$

therefore $\Delta = \mathsf{Adv}^{\overline{\mathsf{VC}}.\mathsf{StaticVerify}}_{\mathcal{A}, \overline{\mathsf{VC}}}(\lambda) \leq \mathsf{Adv}^{\mathsf{VC.StaticVerify}}_{\mathcal{R}, \mathsf{VC}}(\lambda) \leq \mathsf{negl}$ which is a contradiction. Similarly, it is easy to define a reduction $\mathcal{R}$ for an adversary $\mathcal{A}$ for the VC privacy experiments such that $\mathsf{Adv}^{\overline{\mathsf{VC}}.\mathsf{Priv}}_{\mathcal{A}, \overline{\mathsf{VC}}}(\lambda) \leq \mathsf{Adv}^{\mathsf{VC.Priv}}_{\mathcal{R}, \mathsf{VC}}(\lambda)$. □

We complement Fiore-Gennaro's results by providing the proof that their original VC scheme is indeed private, since this is needed to prove the function hiding property of the SFS construction.

Lemma 1. *If CF.*PRF *is a close form efficient PRF, then the Fiore-Gennaro PVC scheme is* private.

Proof: in order to prove the privacy of the Fiore-Gennaro scheme, we define a sequence of games that has the random bit b as input.

- $\mathsf{Game}_1(b, \mathcal{A})$: the experiment $\mathsf{Exp}^{\mathsf{VC.Priv}}_{\mathsf{VC}}(\mathcal{A})$ is executed by using the original Fiore-Gennaro scheme;
- $\mathsf{Game}_2(b, \mathcal{A})$: in this game, the $\widetilde{\rho}_{\mathsf{m}_b}$ value is computed as $\widetilde{\rho}_{\mathsf{m}_b} = e\left(\prod_{i=1}^{l} \mathsf{CF.H}_K(i)^{h_i(\vec{\mathsf{m}_b})}, g_2\right)$;
- $\mathsf{Game}_3(b, \mathcal{A})$: we exchange all the PRF evaluations $\mathsf{CF.H}_K(i)$ with random elements R_i;
- $\mathsf{Game}_4(b, \mathcal{A})$: we split the definition of W into a left and a right component $W = \{(W_{L_i}, W_{R_i})\}_{i=1}^{l} = \{(g_1^{\alpha f_{b_i}}, R_i)\}_{i=1}^{l}$ and we substitute W_i with $W_{L_i} \cdot W_{R_i}$;
- $\mathsf{Game}_5(b, \mathcal{A})$: after the challenge, we compute y which is equal to $f_0(\mathsf{m}_0) = f_1(\mathsf{m}_1)$, define $W_L = g_1^{\alpha \cdot y}$ and then substitute W with just the right component $W = \{W_{R_i}\}_{i=1}^{l}$. The game computes V as $W_L \cdot \prod_{i=1}^{l} R_i^{h_i(\mathsf{m}_b)}$

We highlight the difference between the games in Figure 7 in which we describe the challenger computations made after the challenger bit b sampling and before the bit b' guess. For compactness, we refer to $\mathsf{CF.H}_K$ with just H_K and the notation $\{\cdot\}_i$ where the index i is contained in the set $\{1, \ldots, l\}$.

Claim 1. $\mathsf{Pr}\big(\mathsf{Game}_1(b, \mathcal{A}) = 1\big) = \mathsf{Pr}\big(\mathsf{Game}_2(b, \mathcal{A}) = 1\big)$

Proof: The only difference is on *"how to evaluate"* the $\mathsf{CF.Eval}_{\mathsf{Poly}}$ and by its correctness, the two are equivalent. □

Claim 2. $\big|\mathsf{Pr}\big(\mathsf{Game}_2(b, \mathcal{A}) = 1\big) - \mathsf{Pr}\big(\mathsf{Game}_3(b, \mathcal{A}) = 1\big)\big| \leq \epsilon_{\mathsf{PRF}}$

Proof: The difference between the games is that we replace the evaluation of the PRF with random elements. It is easy to see that an adversary $\mathcal{A}$ able to distinguish between the two games with non-negligible advantage can be used to define an adversary $\mathcal{B}$ able to distinguish the security of the CF.PRF with non-negligible advantage. □

Claim 3. $\mathsf{Pr}\big(\mathsf{Game}_3(b, \mathcal{A}) = 1\big) = \mathsf{Pr}\big(\mathsf{Game}_4(b, \mathcal{A}) = 1\big)$

Proof: The two games are equivalent since there is no difference between the two distributions. □

Claim 4. $\mathsf{Pr}\big(\mathsf{Game}_4(b, \mathcal{A}) = 1\big) = \mathsf{Pr}\big(\mathsf{Game}_5(b, \mathcal{A}) = 1\big)$

$\mathsf{Game}_1(\mathsf{b})$

1 : $\left(K, e\left(g_1, g_2\right)^{\alpha}, \left(f_b, \left\{g_1^{\alpha \cdot f_{b_i}} \mathsf{H}_K(i)\right\}_i\right)\right)$

2 : $\left(\vec{\mathsf{m}}_b, e\left(\mathsf{CF.Eval}_{\mathsf{Poly}}(K, h(\vec{\mathsf{m}}_b)), g_2\right)\right)$

3 : $\left(y, \prod_{i=1}^{l} W_i^{h_i(\vec{\mathsf{m}}_b)}\right)$

$\mathsf{Game}_2(\mathsf{b})$

1 : $\left(K, e\left(g_1, g_2\right)^{\alpha}, \left(f_b, \left\{g_1^{\alpha \cdot f_{b_i}} \mathsf{H}_K(i)\right\}_i\right)\right)$

2 : $\left(\vec{\mathsf{m}}_b, e\left(\prod_{i=1}^{l} \mathsf{H}_K(i)^{h_i(\vec{\mathsf{m}}_b)}, g_2\right)\right)$

3 : $\left(y, \prod_{i=1}^{l} W_i^{h_i(\vec{\mathsf{m}}_b)}\right)$

$\mathsf{Game}_3(\mathsf{b})$

1 : $\left(e\left(g_1, g_2\right)^{\alpha}, \left(f_b, \left\{g_1^{\alpha \cdot f_{b_i}} R_i\right\}_i\right)\right)$

2 : $\left(\vec{\mathsf{m}}_b, e\left(\prod_{i=1}^{l} {R_i}^{h_i(\vec{\mathsf{m}}_b)}, g_2\right)\right)$

3 : $\left(y, \prod_{i=1}^{l} W_i^{h_i(\vec{\mathsf{m}}_b)}\right)$

$\mathsf{Game}_4(\mathsf{b})$

1 : $\left(e\left(g_1, g_2\right)^{\alpha}, (f_b, \left\{\left(g_1^{\alpha \cdot f_{b_i}} R_i\right)\right\}_i)\right)$

2 : $\left(\vec{\mathsf{m}}_b, e\left(\prod_{i=1}^{l} {R_i}^{h_i(\vec{\mathsf{m}}_b)}, g_2\right)\right)$

3 : $\left(y, \prod_{i=1}^{l} (\left(g_1^{\alpha \cdot f_{b_i}}\right) \cdot R_i)^{h_i(\vec{\mathsf{m}}_b)}\right)$

$\mathsf{Game}_5(\mathsf{b} = 1)$

1 : $y = f_1(\mathsf{m}_1)$

2 : $\left(e\left(g_1, g_2\right)^{\alpha}, (f_b, \left\{\left(g_1^{\alpha \cdot f_{b_i}} R_i\right)\right\}_i)\right)$

3 : $\left(\vec{\mathsf{m}}_1, e\left(\prod_{i=1}^{l} {R_i}^{h_i(\vec{\mathsf{m}}_1)}, g_2\right)\right)$

4 : $\left(y, g_1^{\alpha \cdot y} \cdot \prod_{i=1}^{l} {R_i}^{h_i(\vec{\mathsf{m}}_1)}\right)$

$\mathsf{Game}_5(\mathsf{b} = 0)$

1 : $y = f_0(\mathsf{m}_0)$

2 : $\left(e\left(g_1, g_2\right)^{\alpha}, (f_b, \left\{\left(g_1^{\alpha \cdot f_{b_i}} R_i'\right)\right\}_i)\right)$

3 : $\left(\vec{\mathsf{m}}_0, e\left(\prod_{i=1}^{l} {R_i'}^{h_i(\vec{\mathsf{m}}_0)}, g_2\right)\right)$

4 : $\left(y, g_1^{\alpha \cdot y} \cdot \prod_{i=1}^{l} {R_i'}^{h_i(\vec{\mathsf{m}}_0)}\right)$

Fig. 7: The games used for proving the privacy of Fiore-Gennaro PVC scheme.

Proof: The difference between the two games is merely a computational optimisation since $\prod_{i=1}^{l} \left(g_1^{\alpha \cdot f_{b_i}}\right)^{h_i(\mathsf{m}_{b_i})} = g_1^{\alpha \cdot y}$ where $y = f_0(\mathsf{m}_0) = f_1(\mathsf{m}_1)$. Thus, there is no difference between the two games distributions. □

Claim 5. $\mathsf{Pr}\big(\mathsf{Game}_5(1, \mathcal{A}) = 1\big) = \mathsf{Pr}\big(\mathsf{Game}_5(0, \mathcal{A}) = 1\big)$

Proof: in order to prove the equality between the two probabilities, it is important to observe that, since the exponents $h_i(\vec{\mathsf{m}_b})$ and $h_i(\vec{\mathsf{m}_{1-b}})$ are fixed, the probability is measured on the random values R_i and R_i'. Fixed R_i, dually R_i', there exists random values R_i', dually R_i, such that the product $\prod_{i=1}^{l} {R_i}^{h_i(\vec{\mathsf{m}}_b)}$ is equal to $\prod_{i=1}^{l} {R_i'}^{h_i(\vec{\mathsf{m}}_{1-b})}$. Thus, by duality, the probabilities are the same. □

Therefore, the advantage is

$$
\begin{aligned}
&\mathsf{Adv}^{\mathsf{VC.StaticVerify}}_{\mathcal{A},\mathsf{VC}}(\lambda) = \\
&= \left|\mathsf{Pr}\big(\mathsf{Game}_1(1, \mathcal{A}) = 1\big) - \mathsf{Pr}\big(\mathsf{Game}_1(0, \mathcal{A}) = 1\big)\right| \\
&\leq 2 \cdot \sum_{i=1}^{4} \left|\mathsf{Pr}\big(\mathsf{Game}_i(1, \mathcal{A}) = 1\big) - \mathsf{Pr}\big(\mathsf{Game}_{i+1}(1, \mathcal{A}) = 1\big)\right| + \\
&\quad + \left|\mathsf{Pr}\big(\mathsf{Game}_5(1, \mathcal{A}) = 1\big) - \mathsf{Pr}\big(\mathsf{Game}_5(0, \mathcal{A}) = 1\big)\right| \\
&\leq 2 \cdot \epsilon_{\mathsf{PRF}}
\end{aligned}
$$

□

4 Strong Functional Signatures

In this section, we define the *Strong Functional Signature (SFS)* primitive and the related unforgeability and function hiding experiments. We provide a specific SFS instantiation using the variated schemes introduced in Section 3 and prove it achieves unforgeability and function hiding.

4.1 SFS Definition

Our definition of an SFS scheme can be seen as a combination of a PVC and a FS scheme: similar to FS, an SFS scheme achieves delegation of the signing capability *w.r.t.* the master key-pair **and** it also allows the verification of the correct computation of the signing function f through an additional function public key pk_f, as a PVC scheme.

Definition 6 (Strong Functional Signature)**.** *A* Strong Functional Signature (SFS) *scheme for a message space $\mathcal{M}$ and function family $\mathcal{F}$ consists of the PPT algorithms* $\mathsf{SFS} = (\mathsf{SFS.Setup}, \mathsf{SFS.KeyGen}, \mathsf{SFS.Sign}, \mathsf{SFS.Verify})$ *defined as:*

- $\mathsf{SFS.Setup}(\lambda) \rightarrow (\mathsf{msk}, \mathsf{mvk})$: *the setup algorithm takes as input the security parameter λ and outputs the master signing key and the master verification key.*
- $\mathsf{SFS.KeyGen}(\mathsf{msk}, f) \rightarrow (\mathsf{pk}_f, \mathsf{sk}_f)$: *the key generation algorithm takes as input the master signing key and a function $f \in \mathcal{F}$ and outputs a secret signing key sk_f and a public verification key pk_f w.r.t. the function f.*
- $\mathsf{SFS.Sign}(\mathsf{sk}_f, m) \rightarrow (y, \sigma)$: *the signing algorithm takes as input the secret signing key for a function $f \in \mathcal{F}$ and a message in the function domain $m \in \mathcal{D}_f$, and outputs a value $y = f(m)$ and a signature of $f(m)$.*
- $\mathsf{SFS.Verify}(\mathsf{mvk}, \mathsf{pk}_f, y', \sigma) \rightarrow \{0, 1\}$: *the verification algorithm takes as input the master verification key mvk, the public verification key pk_f for the function f, a message y' and a signature σ, and outputs 1 if the signature is valid and a correct computation of f, 0 if it is not a correct computation of f or the signature is not valid.*

We require the following conditions to hold:

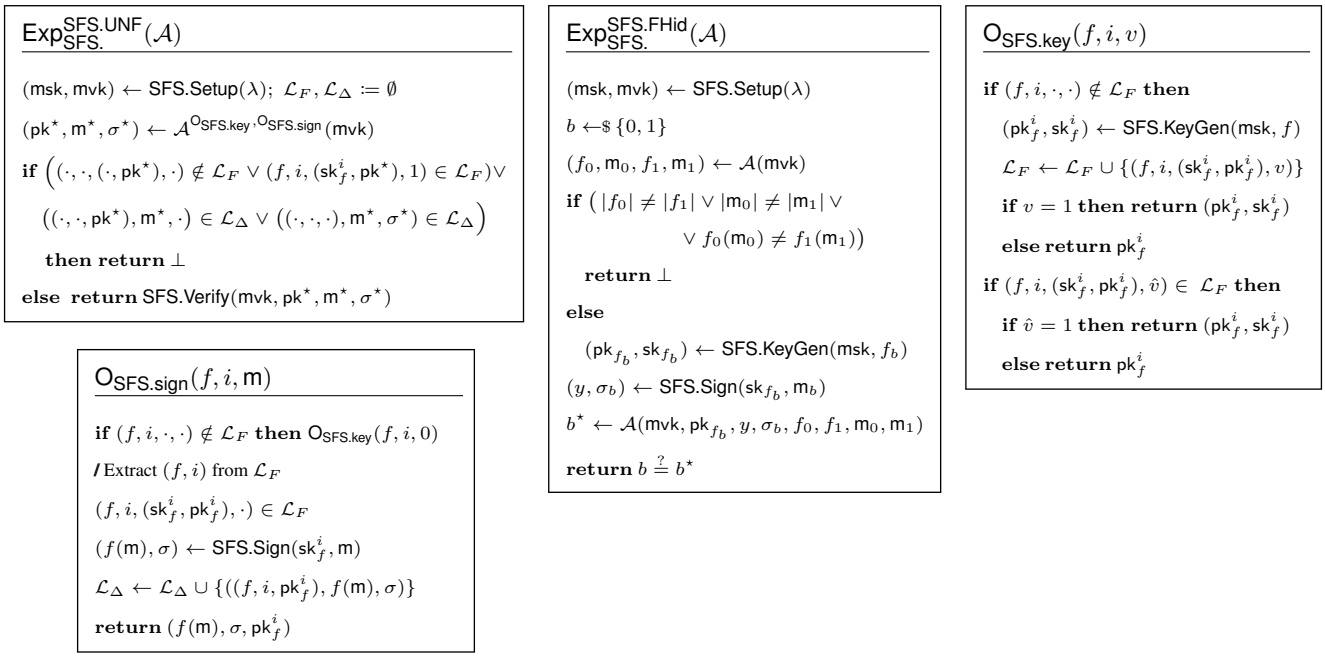

Fig. 8: SFS unforgeability and function hiding experiments.

Correctness: for any function $f \in \mathcal{F}$, for any message $\mathsf{m} \in \mathcal{D}_f$, master keys $(\mathsf{msk},\mathsf{mvk}) \leftarrow \mathsf{SFS.Setup}(\lambda)$, function keys $(\mathsf{pk}_f,\mathsf{sk}_f) \leftarrow \mathsf{SFS.KeyGen}(\mathsf{msk},f)$, and (y,σ) obtained from $\mathsf{SFS.Sign}(\mathsf{sk}_f,\mathsf{m})$, it holds that $\mathsf{SFS.Verify}(\mathsf{mvk},\mathsf{pk}_f,y,\sigma) = 1$.

Succinctness: there exists a polynomial $s(\cdot)$ such that for every $\lambda \in \mathbb{N}$, function $f \in \mathcal{F}$, message $\mathsf{m} \in \mathcal{D}_f$, master keys $(\mathsf{msk},\mathsf{mvk}) \leftarrow \mathsf{SFS.Setup}(\lambda)$, function keys $(\mathsf{pk}_f,\mathsf{sk}_f)$ obtained from $\mathsf{SFS.KeyGen}(\mathsf{msk},f)$, and $(f(\mathsf{m}),\sigma) \leftarrow \mathsf{SFS.Sign}(\mathsf{sk}_f,\mathsf{m})$, it holds with probability 1 that $|\sigma| \leq s(\lambda,|f(\mathsf{m})|)$.

Unforgeability: an SFS scheme is said to be *unforgeable* if the probability of any PPT algorithm $\mathcal{A}$ in the SFS unforgeability experiment $\mathsf{Exp}^{\mathsf{SFS.UNF}}_{\mathsf{SFS}}(\mathcal{A})$ depicted in Fig. 8 to output 1 is negligible. Namely,

$$\mathsf{Adv}^{\mathsf{SFS.UNF}}_{\mathcal{A},\mathsf{SFS}}(\lambda) = \mathsf{Pr}\Big(\mathsf{Exp}^{\mathsf{SFS.UNF}}_{\mathsf{SFS}}(\mathcal{A}) = 1\Big) \leq \mathsf{negl}(\lambda)$$

The main idea behind the unforgeability game is that an adversary $\mathcal{A}$ must present a tamper $(\mathsf{pk}^\star,\mathsf{m}^\star,\sigma^\star)$ for an existing honestly generated public key, whose corresponding secret key is not revealed to $\mathcal{A}$. We allow the adversary to arbitrarily request correct signatures and new key pairs that can be corrupted depending on the value of v, i.e. if $\mathcal{A}$ can obtain a corrupted key pair by querying $\mathsf{O}_{\mathsf{SFS.key}}(f,i,1)$ where $v = 1$. We deliberately **do not allow** $\mathcal{A}$ to corrupt already generated key since this would imply that the third party that generates the function keys is able to identify whenever a specific public key is compromised. Despite being possible in the ideal world, this property is hard to realise in a realistic scenario thus we force $\mathcal{A}$ to declare at the generation, if a key pair is compromised or not.

Function Hiding: an SFS scheme is said to be *function hiding* if the advantage of any PPT algorithm $\mathcal{A}$ in the SFS function hiding experiment $\mathsf{Exp}^{\mathsf{SFS.FHid}}_{\mathsf{SFS}}(\mathcal{A})$, of Figure 8 to output 1 is negligible. Namely,

$$\mathsf{Adv}^{\mathsf{SFS.FHid}}_{\mathcal{A},\mathsf{SFS}}(\lambda) = \left|\mathsf{Pr}\Big(\mathsf{Exp}^{\mathsf{SFS.FHid}}_{\mathsf{SFS}}(\mathcal{A}) = 1\Big) - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$$

Informally, it is impossible for an adversary to distinguish between two different function evaluations and signatures, *i.e.*, given the public verification key of a single function, the adversary cannot infer information on *"what function does the key verify"*.

When comparing to the FS function privacy property, the SFS function hiding requirement might appear *counter-intuitive* since, in the verification phase, it is necessary to use the **public**-key pk_f, which is related to the function f that must be hidden. The SFS function hiding property requires that *"a public-key should just allow the verification of the computation **but** must **not** provide any information of the function"*. This means that from a public-key pk_f, it must be hard to retrieve the corresponding function f.

4.2 An SFS Instantiation

In this subsection, we provide the instantiation of SFS scheme which is a combination of the Fiore-Gennaro's PVC variation (as given in Definition 5) and the BLS variation (as given in Definition 4).

Definition 7. *Let $\overline{\mathsf{BLS}}$ be the variated BLS signature scheme of Definition 4 and $\overline{\mathsf{VC}}$ the variated Fiore-Gennaro PVC scheme of Definition 5. Let the public parameter pp be the description of a bilinear group $(p,g_1,g_2,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e)$ shared between the $\overline{\mathsf{BLS}}$ and the $\overline{\mathsf{VC}}$ schemes. Define the SFS scheme for the polynomial function family $\mathcal{F}$, where every function can be expressed in a binary string representation, with the following algorithms:*

- $\mathsf{SFS.Setup}(\lambda) \rightarrow \mathsf{pp},(\mathsf{msk},\mathsf{mvk})$: *on input the security parameter λ, run $\overline{\mathsf{BLS.Setup}}(\lambda) \rightarrow (\mathsf{MSK},\mathsf{MPK})$, or equivalently $\overline{\mathsf{VC.Setup}}$, and output the master key-pair $(\mathsf{msk},\mathsf{mvk}) = (\mathsf{MSK},\mathsf{MPK})$*
- $\mathsf{SFS.KeyGen}(\mathsf{msk},f) \rightarrow (\mathsf{pk}_f,\mathsf{sk}_f)$: *on input the master secret key msk and a polynomial function f, execute $(\widetilde{\mathsf{sk}}_f,\widetilde{\mathsf{vk}}_f,\widetilde{\mathsf{ek}}_f) \leftarrow \overline{\mathsf{VC.KeyGen}}(\mathsf{pp},\mathsf{msk},f)$, parse the secret key $\widetilde{\mathsf{sk}}_f = (\alpha,g_2^\alpha,K)$ and run the algorithm $(\mathsf{PK}_f,\mathsf{SK}_f) \leftarrow \overline{\mathsf{BLS.KeyGen}}(\lambda,\mathsf{msk},\alpha)$. Output $(\mathsf{pk}_f,\mathsf{sk}_f)$ defined as $\Big((\mathsf{PK}_f,\widetilde{\mathsf{vk}}_f),(\mathsf{SK}_f,(g_2^\alpha,K),\widetilde{\mathsf{ek}}_f)\Big)$*
- $\mathsf{SFS.Sign}(\mathsf{sk}_f,m) \rightarrow (y,\sigma)$: *given as input a secret key sk_f and a message m, parse $\mathsf{sk}_f = \big(\mathsf{SK}_f,(g_2^\alpha,K),\widetilde{\mathsf{ek}}_f\big)$ and execute $(\widetilde{\sigma}_m,\widetilde{\rho}_m) \leftarrow \overline{\mathsf{VC.ProbGen}}((g_2^\alpha,K),m)$, then $\widetilde{\sigma}_y = (y,V) \leftarrow \overline{\mathsf{VC.Compute}}(\widetilde{\mathsf{ek}}_f,\widetilde{\sigma}_m)$ and consequently compute the signature $\ddot{\sigma}_y \leftarrow \overline{\mathsf{BLS.Sign}}(\mathsf{SK}_f,(y,\widetilde{\rho}_m,V))$. Output $(y,\sigma) = \big(y,(\widetilde{\rho}_m,V,\ddot{\sigma}_y)\big)$*

• $\mathit{SFS.Verify}(\mathsf{mvk}, \mathsf{pk}_g, y', \sigma') \to \{0,1\}$: *parse the inputs* $\sigma' = (\widetilde{\rho}_{m'}, V, \ddot{\sigma}_{y'})$ *and* $\mathsf{pk}_g = (\mathsf{PK}_g, \widetilde{\mathsf{vk}}_g)$. *Execute and output:*

$$\bigwedge \begin{array}{c} \overline{\mathit{VC}}.\mathit{Verify}(\mathsf{mvk}, \widetilde{\mathsf{vk}}_g, \widetilde{\rho}_{m'}, (y', V)) \stackrel{?}{=} y' \\ \overline{\mathit{BLS}}.\mathit{Verify}\big(\mathsf{mvk}, \mathsf{PK}_g, (y', \widetilde{\rho}'_m, V), \ddot{\sigma}_{y'}\big) \stackrel{?}{=} 1 \end{array}$$

Correctness: for all $\mathsf{SFS.Setup}(\lambda) \to (\mathsf{msk}, \mathsf{mvk})$, functions $f \in \mathcal{F}$, $\mathsf{SFS.KeyGen}(\mathsf{msk}, f) \to (\mathsf{pk}_f, \mathsf{sk}_f)$ and messages m and $\mathsf{SFS.Sign}(\mathsf{sk}_f, \mathsf{m}) \to (y, \sigma)$, it holds $\mathsf{SFS.Verify}(\mathsf{mvk}, y, \sigma) = 1$ which translates into

$$\bigwedge \begin{array}{c} \overline{\mathsf{VC}}.\mathsf{Verify}(\mathsf{mvk}, \widetilde{\mathsf{vk}}_f, \widetilde{\rho}_\mathsf{m}, (y, V)) \stackrel{?}{=} y \\ \overline{\mathsf{BLS}}.\mathsf{Verify}\big(\mathsf{mvk}, \mathsf{PK}_f, (y, \widetilde{\rho}_\mathsf{m}, V), \ddot{\sigma}_y\big) \stackrel{?}{=} 1 \end{array}$$

and by correctness of the underlying $\overline{\mathsf{BLS}}$ and $\overline{\mathsf{VC}}$ scheme, it is indeed correct.

Succinctness: we observe that the SFS's signature consists of three group elements and it is of constant size, *i.e.* $(\widetilde{\rho}_\mathsf{m}, V, \ddot{\sigma}_y) \in \mathbb{G}_T \times \mathbb{G}_1 \times \mathbb{G}_1$, thereby trivially achieving the succinctness property.

Unforgeability: in order to prove our instantiation to be *unforgeable*, we will prove a reduction from the BLS unforgeability experiment $\mathsf{Exp}^{\mathsf{BLS.UNF}}_{\mathsf{BLS}}(\mathcal{B})$ to the SFS unforgeability experiment $\mathsf{Exp}^{\mathsf{SFS.UNF}}_{\mathsf{SFS}}(\mathcal{A})$.

Theorem 1. *If for all PPT adversaries $\mathcal{B}$ it holds that the advantage $\mathsf{Adv}^{\mathit{BLS.UNF}}_{\mathcal{B},\mathit{BLS}}(\lambda) \le \mathsf{negl}(\lambda)$, then for all PPT adversaries $\mathcal{A}$ it holds $\mathsf{Adv}^{\mathit{SFS.UNF}}_{\mathcal{A},\mathit{SFS}}(\lambda) \le \mathsf{negl}(\lambda)$.*

Proof: assume that there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}^{\mathsf{SFS.UNF}}_{\mathcal{A},\mathsf{SFS}}(\lambda) = \Delta$ for some non-negligible $\Delta > 0$. We construct an adversary $\mathcal{R}$, running $\mathcal{A}$ as a subroutine, to break the unforgeability of the underlying BLS scheme. $\mathcal{R}$ executes $\overline{\mathsf{VC}}.\mathsf{Setup}$ and obtains the master keys $(\mathsf{msk}, \mathsf{mvk})$. $\mathcal{R}$ receives from the BLS challenger the public key PK.

Whenever $\mathcal{A}$ queries a compromised key pair via $\mathsf{O}_{\mathsf{SFS.key}}(f, i, 1)$, $\mathcal{R}$ can generate the keys using $\overline{\mathsf{VC}}.\mathsf{KeyGen}$ and $\overline{\mathsf{BLS}}.\mathsf{KeyGen}$ and therefore can generate keys and compute the signing algorithm and answer to any adversarial signing query. On the other hand, whenever $\mathcal{A}$ queries a uncompromised pair $\mathsf{O}_{\mathsf{SFS.key}}(g, i, 0)$, $\mathcal{R}$ executes $\overline{\mathsf{VC}}.\mathsf{KeyGen}$ and generates the keys $(\widetilde{\mathsf{sk}}_g, \widetilde{\mathsf{ek}}_g, \widetilde{\mathsf{vk}}_g)$. $\mathcal{R}$ samples a random value $z_{(g,i)}$ sets the public key $\mathsf{PK}_2 = \mathsf{PK} \cdot g_2^{z_{(g,i)}}$.

By considering $\mathsf{MSK} = \mathsf{msk}$, $\mathcal{R}$ samples $\alpha, r \in \mathbb{Z}_p$, computes $\mathsf{SK}_1 = g_1^{\mathsf{MSK}+\alpha+r}$ and $\mathsf{PK}_1 = e\left(g_1^{\mathsf{MSK}+\alpha+r}, g_2\right)$ and obtains $\mathsf{PK}_g = (\mathsf{PK}_1, \mathsf{PK}_2)$. Finally, it sends $\mathsf{pk}_g = (\mathsf{PK}_g, \widetilde{\mathsf{vk}}_g)$ to $\mathcal{A}$.

In a nutshell, since the reduction $\mathcal{R}$ can create all the keys **except** the challenged SK, $\mathcal{R}$ is always able to correctly execute the verifiable computation scheme **but** not to sign the final output of a computation of any message m on the **uncompromised** functions g. This means that, whenever $\mathcal{A}$ queries the signing oracle $\mathsf{O}_{\mathsf{SFS.sign}}(g, i, \mathsf{m})$ for an uncompromised function (g, i), $\mathcal{R}$ will sequentially execute $\overline{\mathsf{VC}}.\mathsf{ProbGen}(\widetilde{\mathsf{sk}}_g, \mathsf{m})$ and the algorithm $\overline{\mathsf{VC}}.\mathsf{Compute}(\widetilde{\mathsf{sk}}_g, \widetilde{\sigma}_\mathsf{m})$ to obtain $\widetilde{\sigma}_y = (y, V)$ and $\widetilde{\rho}_\mathsf{m}$. At this point, $\mathcal{R}$ queries the BLS challenger on the message $(y, \widetilde{\rho}_\mathsf{m}, V)$ and obtains $\ddot{\sigma}$ which afterwards modifies into the value $\ddot{\sigma}_y = \mathsf{SK}_1 \cdot \ddot{\sigma} \cdot \mathsf{H}\big((y, \widetilde{\rho}_\mathsf{m}, V)\big)^{z_{(g,i)}}$. $\mathcal{R}$ replies to $\mathcal{A}$ with $(y, (\widetilde{\rho}_\mathsf{m}, V, \ddot{\sigma}_y))$.

Whenever $\mathcal{A}$ outputs the forgery $(\mathsf{pk}^\star, y^\star, \sigma^\star)$, the reduction $\mathcal{R}$ parses the output $\sigma^\star = (\widetilde{\rho}^\star, V^\star, \ddot{\sigma}^\star)$ and outputs the BLS forgery $\left((y^\star, \widetilde{\rho}^\star, V^\star),\ \ddot{\sigma}^\star \cdot \mathsf{SK}_1^{-1} \cdot \mathsf{H}\big((y^\star, \widetilde{\rho}^\star, V^\star)\big)^{-z_{(g,i)}}\right)$. Observe that $\mathcal{A}$ must output a forgery for an uncompromised function that, by construction, is always based on the challenged BLS scheme. The SFS unforgeability experiment's requirements forces $\mathcal{A}$ to always tamper at least one between $(y^\star, \sigma^\star)$ which always translates into $\mathcal{R}$ creating a new tamper never queried before to BLS. Thus, we can conclude that $\Delta = \mathsf{Adv}^{\mathsf{SFS.UNF}}_{\mathcal{A},\mathsf{SFS}}(\lambda) \le \mathsf{Adv}^{\mathsf{BLS.UNF}}_{\mathcal{B},\mathsf{BLS}}(\lambda)$ which is a contradiction. □

Remark 3. *The unforgeability experiment $\mathit{Exp}^{\mathit{SFS.UNF}}_{\mathit{SFS}}(\mathcal{A})$ requires the adversary $\mathcal{A}$ to provide a tamper for a challenged public key $\mathsf{pk}^\star$ of a function g which* must exist *and be* uncompromised. *This means that $\mathcal{A}$ queried $\mathsf{O}_{\mathit{SFS.key}}(g, *, 0)$ explicitly or implicitly via the signing oracle, and only owns the public key $\mathsf{pk}^\star$.*

As a matter of curiosity, Theorem 1's proof can be interpreted as the case where $\mathcal{A}$ cannot forge even if the secret keys are partially *compromised. In particular, consider that the proof's reduction $\mathcal{R}$ returns to $\mathcal{A}$ **all** the $\overline{\mathit{VC}}.\mathit{KeyGen}$ generated keys $(\widetilde{\mathsf{sk}}_g, \widetilde{\mathit{ek}}_g, \widetilde{\mathsf{vk}}_g)$ which would allow $\mathcal{A}$ to* always *pass the verification $\overline{\mathit{VC}}.\mathit{Verify}$. Despite this additional concession, the proof shows that $\mathcal{A}$ is still unable to provide a tamper for $\overline{\mathit{BLS}}.$ since $\mathcal{A}$ does not hold the $\overline{\mathit{BLS}}.$ signing secret key, thus making it impossible to create a* SFS *tamper.*

Function Hiding: in order to prove our instantiation to be *function hiding*, we will show a reduction from the VC function privacy experiment $\mathsf{Exp}^{\mathsf{VC.Priv}}_{\mathsf{VC}}(\mathcal{B})$ to the SFS function hiding experiment $\mathsf{Exp}^{\mathsf{SFS.FHid}}_{\mathsf{SFS}}(\mathcal{A})$.

Theorem 2. *If for all PPT adversaries $\mathcal{B}$ it holds that the advantage $\mathsf{Adv}^{\mathit{VC.Priv}}_{\mathcal{B},\mathit{VC}}(\lambda) \le \mathsf{negl}(\lambda)$, then for all PPT adversaries $\mathcal{A}$ it holds $\mathsf{Adv}^{\mathit{SFS.FHid}}_{\mathcal{A},\mathit{SFS}}(\lambda) \le \mathsf{negl}(\lambda)$.*

Proof: assume the existence of a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}^{\mathsf{SFS.FHid}}_{\mathcal{A},\mathsf{SFS}}(\lambda) = \Delta$ for some non-negligible $\Delta > 0$. We then construct an adversary $\mathcal{B}$, running $\mathcal{A}$ as a subroutine, to break the privacy security of the underlying VC scheme. Let $\mathcal{R}$ be the reduction from the $\mathsf{VC.Priv}$ experiment to the $\mathsf{SFS.FHid}$ one and therefore $\mathcal{B}$ the final adversary that uses $\mathcal{R}$ and $\mathcal{A}$. $\mathcal{R}$ execute $\overline{\mathsf{VC}}.\mathsf{Setup}(\lambda) \to (\widetilde{\mathsf{msk}}, \widetilde{\mathsf{mpk}})$ and sends $\mathsf{mvk} = \widetilde{\mathsf{mpk}}$ to the SFS adversary $\mathcal{A}$. $\mathcal{A}$ replies with the challenge $(f_0, \mathsf{m}_0, f_1, \mathsf{m}_1)$ which is forwarded to the $\mathsf{VC.Priv}$ challenger by $\mathcal{R}$. $\mathcal{R}$ receives $(\widetilde{\mathsf{vk}}_{f_b}, \widetilde{\sigma}_{y_b}, \widetilde{\rho}_b)$ where $\widetilde{\sigma}_{y_b} = (y, V_b)$ with y which is equal to $f_0(\mathsf{m}_0) = f_1(\mathsf{m}_1)$. $\mathcal{R}$ executes $\overline{\mathsf{BLS}}.\mathsf{KeyGen}(\lambda, \mathsf{msk}, \alpha)$ for some random $\alpha \in \mathbb{G}$ and obtain $\mathsf{SK} = (\mathsf{SK}_1, \mathsf{SK}_2) = (g_1^{\mathsf{msk}+\alpha+r}, k)$ and $\mathsf{PK} = (\mathsf{PK}_1, \mathsf{PK}_2) = (e(g_1, g_2)^\alpha, g_2^k)$, then it signs $\overline{\mathsf{BLS}}.\mathsf{Sign}(\mathsf{SK}, y)$ and obtains $\ddot{\sigma}$. The reduction $\mathcal{R}$ then replies to the $\mathcal{A}$ with the tuple $(\widetilde{\mathsf{vk}}_{f_b}, \widetilde{\sigma}^\star_{y_b}, \widetilde{\rho}_b, \mathsf{PK}, \ddot{\sigma})$ where $\widetilde{\sigma}^\star_{y_b} = (y, V_b{}^\star)$ which is equal to $(y, V_b \cdot g_1^{\mathsf{msk}\cdot y})$. Finally, $\mathcal{A}$'s guess is just forwarded to the challenger in VC's privacy game.

By observing the $\mathsf{SFS.Verify}$ algorithm, we get

$$\bigwedge \begin{array}{c} \overline{\mathsf{VC}}.\mathsf{Verify}(\mathsf{mvk}, \widetilde{\mathsf{vk}}_{f_b}, \widetilde{\rho}_{\mathsf{m}_b}, (y, V_b{}^\star)) \stackrel{?}{=} y \\ \overline{\mathsf{BLS}}.\mathsf{Verify}(\mathsf{mvk}, \mathsf{PK}, \overline{\mathsf{BLS}}.\mathsf{Sign}(\mathsf{SK}, y)) \stackrel{?}{=} 1 \end{array}$$

and since the right side is always true, the left side is equivalent to

$$\begin{aligned} &\overline{\mathsf{VC}}.\mathsf{Verify}(\mathsf{mvk}, \widetilde{\mathsf{vk}}_{f_b}, \widetilde{\rho}_{\mathsf{m}_b}, (y, V_b{}^\star)) \iff \\ &\iff e\left(V_b{}^\star, g_2\right) \stackrel{?}{=} (\mathsf{mvk} \cdot \widetilde{\mathsf{vk}}_{f_b})^y \cdot \widetilde{\rho}_{\mathsf{m}_b} \\ &\iff e\left(V_b \cdot g_1^{\mathsf{msk}\cdot y}, g_2\right) \stackrel{?}{=} \mathsf{mvk}^y \cdot \widetilde{\mathsf{vk}}_{f_b}^y \cdot \widetilde{\rho}_{\mathsf{m}_b} \\ &\iff \mathsf{mvk}^y \cdot e(V_b, g_2) \stackrel{?}{=} \mathsf{mvk}^y \cdot \widetilde{\mathsf{vk}}_{f_b}^y \cdot \widetilde{\rho}_{\mathsf{m}_b} \\ &\iff e(V_b, g_2) \stackrel{?}{=} \widetilde{\mathsf{vk}}_{f_b}^y \cdot \widetilde{\rho}_{\mathsf{m}_b} \\ &\iff \mathsf{VC.Verify}(\widetilde{\mathsf{vk}}_{f_b}, \widetilde{\rho}_{\mathsf{m}_b}, (y, V_b)) \end{aligned}$$

Therefore, if the adversary $\mathcal{A}$ has an advantage Δ, the built adversary $\mathcal{B}$ for $\mathsf{VC.Priv}$ that uses $\mathcal{R}$ has advantage Δ. In other word, we conclude that $\Delta = \mathsf{Adv}^{\mathsf{SFS.FHid}}_{\mathcal{A},\mathsf{SFS}}(\lambda) \le \mathsf{Adv}^{\mathsf{VC.Priv}}_{\mathcal{B},\mathsf{VC}}(\lambda)$ which is a contradiction. □

5 Conclusion

Verifying the correctness of computations is a very valuable property considering the ever-increasing cloud-assisted computing paradigm. This paper defines *Strong Functional Signature (SFS)* as an enhanced version of functional signatures with verifiable computation properties. In a nutshell, SFS introduce a functional public key pk_f that works as a commitment for a function f. This public-key allows in verification to guarantee the correct computation of the committed function without revealing any information on the function **and** to distinguish between different computed functions in a privacy-preserving way. Furthermore, we provide a concrete instantiation of an SFS scheme and prove that it satisfies the properties of *unforgeability* and *function hiding*.

Acknowledgment

This work was partially supported by the Swedish Research Council (Vetenskapsrådet) through the grant PRECIS (621-2014-4845).

6 References

1 Michael Backes, Sebastian Meiser, and Dominique Schröder. Delegatable functional signatures. In *Public-Key Cryptography–PKC 2016*, pages 357–386. Springer, 2016.

2 Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. In *International Workshop on Public Key Cryptography*, pages 520–537. Springer, 2014.

3 Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012.

4 Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 111–120. ACM, 2013.

5 Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

6 Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional Signatures and Pseudorandom Functions. In Hugo Krawczyk, editor, *Public-Key Cryptography – PKC 2014*, Lecture Notes in Computer Science, pages 501–519. Springer Berlin Heidelberg, 2014.

7 W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006.

8 Dario Fiore and Rosario Gennaro. Publicly Verifiable Delegation of Large Polynomials and Matrix Computations, with Applications. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 501–512, New York, NY, USA, 2012. ACM.

9 Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently Verifiable Computation on Encrypted Data. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 844–855, New York, NY, USA, 2014. ACM.

10 Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Annual Cryptology Conference*, pages 465–482. Springer, 2010.

11 Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 99–108. ACM, 2011.

12 Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

13 Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *Theory of Cryptography*, pages 222–242. Springer, 2013.

14 Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography Conference*, pages 422–439. Springer, 2012.