Anonymous Single-Round Server-Aided Verification

Elena Pagnin[1](✉), Aikaterini Mitrokotsa[1], and Keisuke Tanaka[2]

[1] Chalmers University of Technology, Gothenburg, Sweden
{elenap, aikmitr}@chalmers.se
[2] Tokyo Institute of Technology, Japan
keisuke@is.titech.ac.jp

Abstract. Server-Aided Verification (SAV) is a method that can be employed to speed up the process of verifying signatures by letting the verifier outsource part of its computation load to a third party. Achieving fast and reliable verification under the presence of an untrusted server is an attractive goal in cloud computing and internet of things scenarios.
In this paper, we describe a simple framework for SAV where the interaction between a verifier and an untrusted server happens via a single-round protocol. We propose a security model for SAV that refines existing ones and includes the new notions of SAV-*anonymity* and *extended unforgeability*. In addition, we apply our definitional framework to provide the first generic transformation from any signature scheme to a single-round SAV scheme that incorporates verifiable computation. Our compiler identifies two independent ways to achieve SAV-anonymity: *computationally*, through the privacy of the verifiable computation scheme, or *unconditionally*, through the adaptibility of the signature scheme.
Finally, we define three novel instantiations of SAV schemes obtained through our compiler. Compared to previous works, our proposals are the only ones which simultaneously achieve existential unforgeability and soundness against collusion.

Keywords. Server-Aided Verification, Digital Signatures, Anonymity, Verifiable Computation.

1 Introduction

The design of new efficient and secure signature schemes is often a challenging task, especially when the target devices on which the scheme should run have *limited resources*, as it happens in the Internet of Things (IoT). Nowadays many IoT devices can perform quite *expensive* computations. For instance, smartphones have gained significant computational power. Carrying out several expensive tasks, however, leads to undesirable consequences as, *e.g.*, draining the battery of the device [11]. We consider signed auctions as a motivating example in an IoT setting. In signed auctions, bidders sign their offers to guarantee that the amount is correct and that the offer belongs to them. The auctioneer considers a bid valid only if its signature is verified. Imagine that the auctioneer checks

the validity of the bids using a resource-limited device. In this case, running the signature verification algorithm several times drastically affects the device's performance. In this setting one may wonder:

> Can an auctioneer *efficiently*, *securely* and *privately* check the authenticity of signed bids using a *resource-limited* device?

This paper addresses the above question in case the auctioneer has access to a computationally powerful, yet untrusted, server. This is indeed the setting of server-aided verification.

1.1 Previous Work

The concept of Server-Aided Verification (SAV) was introduced in the nineties in two independent works [1,18], and refined for the case of signature and authentication schemes by Girault and Lefranc [15]. The aim of SAV is to guarantee security and reliability of the outcome of a verification procedure when part of the computation is offloaded from a trusted device, called the verifier, to an untrusted one, the server.

All existing security models for SAV consider existential forgery attacks, where the adversary, *i.e.*, the malicious server, tries to convince the verifier that an invalid signature is valid [8,15,21,23,25,26]. Despite the fundamental theoretical contributions, [15] did not consider attack scenarios in which the malicious signer colludes with the server, *e.g.*, by getting control over the server, in order to tamper with the outcome of the server-aided verification of a signature. The so-called collusion attack was defined by Wu *et al.* in [25,26] together with two SAV schemes claimed to be collusion-resistant. Subsequent works revisited the notion of signer-server collusion [8,22,23]. The most complete and rigorous definition of collusion attack is due to Chow *et al.* [8], who also showed that the protocols in [26] are no longer collusion resistant under the new definition [7,8]. Recently, Cao *et al.* [7] rose new concerns about the artificiality and the expensive communication costs of the SAV in [25].

Chow *et al.* [8] showed that the enabler of many attacks against SAV is the absence of an integrity check on the results returned by the server. Integrity however, is not the only concern when outsourcing computations. In this paper, we address for the first time privacy concerns and we introduce the notion of anonymity in the context of SAV of signatures.

1.2 Contributions

The main motivation of this work is the need for formal and realistic definitions in the area of server-aided verification. To this purpose we:

- Introduce a formalism which allows for an intuitive description of *single-round* SAV signature schemes (Section 3);

- Define a security model that includes three new security notions: SAV-*anonymity* (Section 4.3), *extended* existential unforgeability and extended *strong* unforgeability (Section 4.1);

- Describe the *first compiler* to a SAV signature scheme from any signature and a verifiable computation scheme (Section 5). Besides its simplicity, our generic composition identifies *sufficient requirements* on the underlying primitives to achieve security. In particular, we prove that under mild assumptions our compiler provides: extended (existential/strong) unforgeability (Theorem 1); soundness against collusion (Theorem 2); and SAV-anonymity when either the employed verifiable computation is private (Theorem 3) or the signature scheme is adaptive (Theorem 4).

- Apply our generic composition to obtain *new* SAV schemes for the BLS signature [3] (Section 6.1), Waters' signature Wat [24] (Section 6.2) and the *first* SAV for the CL signature by Camenisch and Lysyanskaya [5] (Section 6.3). While preserving efficiency, our proposals achieve better security than previous works (Table 1).

2 Preliminaries

Throughout the paper, $x \leftarrow A(y)$ denotes the output x of an algorithm A run with input y. If X is a finite set, by $x \xleftarrow{R} X$ we mean x is sampled from the uniform distribution over the set X. The expression $cost(A)$ refers to the computational cost of running algorithm A. For any positive integer n, $[n] = \{1, \ldots, n\}$ and $\mathbb{G}_n$ is a group of order n. A function $f : \mathbb{N} \to \mathbb{R}$ is said to be *negligible* if $f(n) < 1/poly(n)$ for any polynomial $poly(\cdot)$ and any $n > n_0$, for suitable $n_0 \in \mathbb{N}$. Finally, ε denotes a negligible function.

2.1 Signature schemes

Signature schemes [4,5,13] enable one to sign a message in such a way that anyone can verify the signature and be convinced that the message was created by the signer. Formally,

Definition 1 (Signature scheme). *A signature scheme* $\Sigma = (\mathsf{SetUp}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *consists of four, possibly randomized, polynomial time algorithms where:*

- $\mathsf{SetUp}(1^\lambda)$*: on input the security parameter* $\lambda \in \mathbb{N}$*, the setup algorithm returns the global parameters* gp *of the scheme, which include a description of the message and the signature spaces* $\mathcal{M}$*,* $\mathcal{S}$*. The* gp *are input to all the following algorithms, even when not specified.*
- $\mathsf{KeyGen}()$*: the key generation outputs public-secret key pairs* $(\mathsf{pk}, \mathsf{sk})$*.*
- $\mathsf{Sign}(\mathsf{sk}, m)$*: on input a secret key* sk *and a message* $m \in \mathcal{M}$*, the sign algorithm outputs a signature* $\sigma \in \mathcal{S}$ *for* m*.*
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$*: The verification algorithm is a deterministic algorithm that given a public key* pk*, a message* $m \in \mathcal{M}$ *and a signature* $\sigma \in \mathcal{S}$*, outputs* $\mathsf{b} = 1$ *for acceptance, or* $\mathsf{b} = 0$ *for rejection.*

Definition 2 ((In)Valid signatures). *Let* Σ *be a signature scheme. We say that a signature* $\sigma \in \mathcal{S}$ *is valid for a message* $m \in \mathcal{M}$ *under the key* pk *if* $\mathsf{Verify}(\mathsf{pk}, m, \sigma) = 1$*. Otherwise, we say that* σ *is invalid.*

In this paper, we refer to (in)valid *signatures* also as (in)valid message-signature *pairs*.

2.2 Verifiable computation

Verifiable computation schemes enable a client to delegate computations to one or more untrusted servers, in such a way that one can efficiently verify the correctness of the result returned by the server [2,12]. Gennaro *et al.* [14] formalised private verification of outsourced computations as:

Definition 3 (Verifiable Computation scheme [14]). *A verifiable computation scheme* $\Gamma = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Comp}, \mathsf{Verify})$ *consists of four possibly randomized algorithms where:*

$\mathsf{KeyGen}(\lambda, f)$**:** *given the security parameter* λ *and a function* f*, the key generation algorithm produces a public key* pk*, that encodes the target function* f*, and a secret key* sk*.*

$\mathsf{ProbGen}(\mathsf{sk}, x)$**:** *given the secret key* sk *and the input data* x*, the problem generation algorithm outputs a public value* ω_x *and a private value* τ_x*.*

$\mathsf{Comp}(\mathsf{pk}, \omega_x)$**:** *given the public key* pk *and the encoded input* ω_x*, this algorithm computes* ω_y*, which is an encoding of* $y = f(x)$*.*

$\mathsf{Verify}(\mathsf{sk}, \tau_x, \omega_y)$**:** *given* sk*,* τ_x *and the encoded result* ω_y*, the verification algorithm returns* y *if* ω_y *is a valid encoding of* $f(x)$*, and* $\perp$ *otherwise.*

A verifiable scheme is efficient if verifying the outsourced computation requires less computational effort than computing the function f on the data x, *i.e.*, $cost(\mathsf{ProbGen}) + cost(\mathsf{Verify}) < cost(f(x))$.

In the remainder of the paper, we often drop the indexes and write $\tau_x = \tau$, $\omega_x = \omega$, $\omega_y = \rho$ and denote by y the output of $\mathsf{Verify}(\mathsf{sk}, \tau, \rho)$.

3 Single-round server-aided verification

In the context of signatures, server-aided verification is a method to improve the efficiency of a resource-limited verifier by outsourcing part of the computation load required in the signature verification to a computationally powerful server. Intuitively SAV equips a signature scheme with:

- An additional $\mathsf{SAV.VSetup}$ algorithm that sets up the server-aided verification and outputs a public component pb (given to the server) and a private one pr (held by the verifier only). [3]
- An interactive protocol $\mathsf{AidedVerify}$ executed between the verifier and the server that outputs: 0 if the input signature is invalid; 1 if the input signature is valid; and $\perp$ otherwise, *e.g.*, when the server returns values that do not match the expected output of the outsourced computation.

In this work, we want to reduce the communication cost of $\mathsf{AidedVerify}$ and restrict this to a single-round (two-message) interactive protocol. This choice enables us to describe the $\mathsf{AidedVerify}$ protocol as a sequence of three algorithms: $\mathsf{SAV.ProbGen}$ (run by the verifier), $\mathsf{SAV.Comp}$ (run by the server) and $\mathsf{SAV.Verify}$

[3] In [8,26] the output of $\mathsf{SAV.VSetup}$ is called **Vstring**.

(run by the verifier). This limitation is less restrictive than it may appear: all the instantiations of SAV signature schemes in [15,17,21,23,25,26,28] are actually single-round SAV.

We define single-round server-aided verification signature schemes as:

Definition 4 (SAV). *A single-round server-aided verification signature scheme is defined by the following possible randomized algorithms:*

$\mathsf{SAV.Init}(1^\lambda)$**:** *on input the security parameter $\lambda \in \mathbb{N}$, the initialisation algorithm returns the global parameters* gp *of the scheme, which are input to all the following algorithms, even when not specified.*

$\mathsf{SAV.KeyGen}()$**:** *the key generation algorithm outputs a secret key* sk *(used to sign messages) and the corresponding public key* pk*.*

$\mathsf{SAV.VSetup}()$**:** *the server-aided verification setup algorithm outputs a public verification-key* pb *and a private one* pr*.*

$\mathsf{SAV.Sign}(\mathsf{sk}, m)$**:** *given a secret key* sk *and a message m the sign algorithm produces a signature σ.*

$\mathsf{SAV.ProbGen}(\mathsf{pr}, \mathsf{pk}, m, \sigma)$ **:** *on input the private verification key* pr*, the public key* pk*, a message m and a signature σ, this algorithm outputs a public-private data pair (ω, τ) for the server-aided verification.*

$\mathsf{SAV.Comp}(\mathsf{pb}, \omega)$**:** *on input the public verification key* pb *and ω the outsourced-computation algorithm returns ρ.*

$\mathsf{SAV.Verify}(\mathsf{pr}, \mathsf{pk}, m, \sigma, \rho, \tau)$**:** *the verification algorithm takes as input the private verification-key* pr*, the public key* pk*, m, σ, ρ and τ. The output is $\Delta \in \{0, 1, \perp\}$.*

Intuitively, the output Δ of SAV.Verify has the following meanings:

- $\Delta = 1$: the pair (m, σ) is considered valid and we say that (m, σ) *verifies in the server-aided sense*;
- $\Delta = 0$: the pair (m, σ) is considered invalid and we say that (m, σ) does *not verify in the server-aided sense*;
- $\Delta = \perp$: the server-aided verification has failed, ρ is rejected (not σ), and nothing is inferred about the validity of (m, σ).

Unless stated otherwise, from now on SAV refers to a single-round server-aided signature verification scheme as in Definition 4. Definition 4 implicitly allows to delegate the computation of several inputs, as long as all inputs can be sent in a single round, as a vector ω.

Completeness and efficiency of SAV are defined as follows.

Definition 5 (SAV completeness). *A* SAV *is said to be complete if for all $\lambda \in \mathbb{N}$, $\mathsf{gp} \leftarrow \mathsf{SAV.Init}(1^\lambda)$, for any pair of keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SAV.KeyGen}()$, $(\mathsf{pb}, \mathsf{pr}) \leftarrow \mathsf{SAV.VSetup}()$ and message $m \xleftarrow{R} \mathcal{M}$; given $\sigma \leftarrow \mathsf{SAV.Sign}(\mathsf{sk}, m)$, $(\omega, \tau) \leftarrow \mathsf{SAV.ProbGen}(\mathsf{pr}, \mathsf{pk}, m, \sigma)$ and $\rho \leftarrow \mathsf{SAV.Comp}(\mathsf{pb}, \omega)$, it holds:*

$$\mathsf{Prob}\left[\mathsf{SAV.Verify}(\mathsf{pr}, \mathsf{pk}, m, \sigma, \rho, \tau) = 1\right] > 1 - \varepsilon$$

where the probability is taken over the coin tosses of SAV.Sign*,* SAV.ProbGen*.*

Definition 6 (SAV efficiency). *A* SAV *for a signature scheme* $\Sigma = (\mathsf{SetUp}_\Sigma, \mathsf{KeyGen}_\Sigma, \mathsf{Sign}_\Sigma, \mathsf{Verify}_\Sigma)$ *is said to be efficient if the computational cost of the whole server-aided verification is less than the cost of running the standard signature verification, i.e.,*

$$\big(cost(\mathsf{SAV.ProbGen}) + cost(\mathsf{SAV.Verify})\big) < cost(\mathsf{Verify}_\Sigma)\ .$$

4 Security model

In server-aided verification there are two kinds of adversaries to be considered: the one that solely controls the server used for the aided-verification, and the one that additionally knows the secret key for signing (signer-server collusion). In the first case, we are mostly concerned about forgeries against the signature scheme, while in the second scenario we want to avoid some kind of repudiation [7]. Existing security models for SAV consider existential unforgeability (EUF) and soundness against collusion (SAC) [8,26]. In this section, we extend the notion of EUF to capture new realistic attack scenarios and we consider for the first time signer anonymity in server-aided verification.

In what follows, the adversary $\mathcal{A}$ is a probabilistic polynomial time algorithm. We denote by q_s (*resp.* q_v) the upper bound on the number of signature (*resp.* verification) queries in each query phase.

4.1 Unforgeability

Intuitively, a SAV signature scheme is unforgeable if a malicious server, taking part to the server-aided verification process, is not able to tamper with the output of the protocol. All the unforgeability notions presented in this section are based on the unforgeability under chosen message and verification attack (UF-ACMV) experiment:

Definition 7. *The unforgeability under chosen message and verification experiment* ($\mathbf{Exp}_{\mathcal{A}}^{\mathsf{UF\text{-}ACMV}}[\lambda]$) *goes as follows:*

Setup. *The challenger* $\mathcal{C}$ *runs the algorithms* SAV.Init, SAV.KeyGen *and* SAV.VSetup *to obtain the system parameters* gp, *the key pair* (pk, sk), *and the public-private verification keys* (pb, pr). *The adversary* $\mathcal{A}$ *is given* pk, pb, *while* sk *and* pr *are withheld from* $\mathcal{A}$.

Query Phase I. *The adversary can make a series of queries which may be of the following two kinds:*

- *sign: $\mathcal{A}$ chooses a message m and sends it to $\mathcal{C}$. The challenger behaves as a signing oracle: it returns the value $\sigma \leftarrow \mathsf{SAV.Sign}(\mathsf{sk}, m)$ and stores the pair (m, σ) in an initially empty list $L \subset \mathcal{M} \times \mathcal{S}$.*

- *verify: $\mathcal{A}$ begins the interactive (single-round) protocol for server-aided verification by supplying a message-signature pair (m, σ) to its challenger. $\mathcal{C}$ simulates a verification oracle: it runs $\mathsf{SAV.ProbGen}(\mathsf{pr}, \mathsf{pk}, m, \sigma) \to (\omega, \tau)$, returns ω to $\mathcal{A}$, and waits for a second input. Upon receiving an answer ρ from the adversary, the challenger returns $\Delta \leftarrow \mathsf{SAV.Verify}(\mathsf{pr}, \mathsf{pk}, m, \sigma, \rho, \tau)$.*

The adversary can choose its queries adaptively based on the responses to previous queries, and can interact with both oracles at the same time.

Challenge. *$\mathcal{A}$ chooses a message-signature pair (m^*, σ^*) and sends it to $\mathcal{C}$. The challenger computes $(\hat{\omega}, \hat{\tau}) \leftarrow \mathsf{SAV.ProbGen}(\mathsf{pr}, \mathsf{pk}, m^*, \sigma^*)$. The value $\hat{\tau}$ is stored and withheld from $\mathcal{A}$, while $\hat{\omega}$ is sent to the adversary.*

Query Phase II. *In the second query phase the sign queries are as before, while the verify queries are answered using the same $\hat{\tau}$ generated for the challenge, i.e., $\mathcal{A}$ submits only ρ and $\mathcal{C}$ replies with $\Delta \leftarrow \mathsf{SAV.Verify}(\mathsf{pr}, \mathsf{pk}, m^*, \sigma^*, \rho, \hat{\tau})$.*

Forgery. *$\mathcal{A}$ outputs the tuple (m^*, σ^*, ρ^*).*

The experiment outputs 1 if (m^*, σ^*, ρ^*) is a forgery (see Definition 8), and 0 otherwise.

Unlike unforgeability for digital signatures, in SAV the adversary can influence the outcome of the signature verification through the value ρ^*. Moreover, $\mathcal{A}$ can perform verification queries. This is a crucial requirement as the adversary cannot run SAV.Verify on its own, since pr and τ are withheld from $\mathcal{A}$. In practice, whenever the output of the server-aided verification is $\perp$ the verifier could abort and stop interacting with the malicious server. In this work, we ignore this case and follow the approach used in [8] and in verifiable computation [14] where the adversary 'keeps on querying' independently of the outcome of the verification queries.

Definition 8 (Forgery). *Consider an execution of the* UF-ACMV *experiment where (m^*, σ^*, ρ^*) is the tuple output by the adversary. We define three types of forgery:*

type-1a forgery*: $(m^*, \cdot) \notin L$ and $1 \leftarrow \mathsf{SAV.Verify}(\mathsf{pr}, \mathsf{pk}, m^*, \sigma^*, \rho^*, \hat{\tau})$.*

type-1b forgery*: $(m^*, \sigma^*) \notin L$ and $1 \leftarrow \mathsf{SAV.Verify}(\mathsf{pr}, \mathsf{pk}, m^*, \sigma^*, \rho^*, \hat{\tau})$.*

type-2 forgery*: $(m^*, \sigma^*) \in L$ and $0 \leftarrow \mathsf{SAV.Verify}(\mathsf{pr}, \mathsf{pk}, m^*, \sigma^*, \rho^*, \hat{\tau})$,*

Existential unforgeability for SAV signature schemes is defined for a quite weak adversary: the second query phase is skipped and only type-1a forgeries are considered:

Definition 9 (Existential Unforgeability (EUF) [8]). *A SAV scheme is (ε, q_s, q_v)-existentially unforgeable under adaptive chosen message and verification attacks if $\mathsf{Prob}\left[\mathbf{Exp}_{\mathcal{A}}^{\text{UF-ACMV}}[\lambda] = 1\right] < \varepsilon$ and the experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{UF-ACMV}}[\lambda]$ outputs 1 only on **type-1a forgeries**, and no query is performed in the **Query Phase II**.*

This notion of unforgeability fails to capture some realistic attack scenarios. For instance, consider the case of signed auctions. The adversary is a bidder and wants to keep the price of the goods he is bidding on under a certain threshold. A simple way to achieve this goal is to get control over the server used for the SAV and prevent signatures of higher bids from verifying correctly. This motivates us to *extend* the notion of EUF in [8,26] to also account for malicious servers tampering with the verification outcome of honestly generated message-signature pairs:

Definition 10 (Extended Existential Unforgeability (ExEUF)). *A SAV scheme is (ε, q_s, q_v)-extended existentially unforgeable under adaptive chosen message and verification attacks if* $\mathsf{Prob}\left[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{UF\text{-}ACMV}}[\lambda] = 1\right] < \varepsilon$ *and the experiment* $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{UF\text{-}ACMV}}[\lambda]$ *outputs 1 on* ***type-1a*** *and* ***type-2 forgeries****.*

Extended existential unforgeablility deals with a stronger adversary than the one considered in EUF: in ExEUF the adversary can perform two different types of forgeries and has access to an additional query phase (after setting the challenge). Resembling the notion of the strongly unforgeable signatures [4], we introduce *extended strong* unforgeability for SAV:

Definition 11 (Extended Strong Unforgeability (ExSUF)). *A SAV scheme is (ε, q_s, q_v)-extended strong unforgeable under adaptive chosen message and verification attacks if* $\mathsf{Prob}\left[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{UF\text{-}ACMV}}[\lambda] = 1\right] < \varepsilon$ *and* $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{UF\text{-}ACMV}}[\lambda]$ *outputs 1 on* ***type-1a, type-1b*** *and* ***type-2 forgeries****.*

In ExSUF there is no restriction on the pair (m^*, σ^*) chosen by the adversary: it can be a new message (type-1a), a new signature on a previously-queried message (type-1b) or an honestly generated pair obtained in the first Query Phase (type-2).

4.2 Soundness against collusion

In collusion attacks, the adversary controls the server used for the aided verification and holds the signer's secret key. This may happen when a malicious signer hacks the server and wants to tamper with the outcome of a signature verification. As a motivating example consider signed auctions. The owner of a good could take part to the auction (as the malicious signer) and influence its price. For instance, in order to increase the cost of the good, the malicious signer can produce an invalid signature for a high bid (message) and make other bidders overpay for it. To tamper with the verification of the invalid signature, the malicious signer can use the server and make his (invalid) signature verify when the bid is stated. However, in case no one outbids him, the malicious signer can repudiate the signature as it is actually invalid.

We define collusion as in [8], with two minor adaptations: (i) we use our single-round framework, that allows us to clearly state the information flow between $\mathcal{A}$ and $\mathcal{C}$; and (ii) we introduce a second query phase, after the challenge phase (to strengthen the adversary).

Definition 12 (Soundness Against Collusion (SAC)). *Define the experiment* $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{ACVAuC}}[\lambda]$ *to be* $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{UF\text{-}ACMV}}[\lambda]$ *where:*

- in the ***Setup*** *phase, $\mathcal{C}$ gives to $\mathcal{A}$ all keys except* pr*, and*

- no sign query is performed, and

- the tuple (m^*, σ^*, ρ^*) output by $\mathcal{A}$ at the end of the experiment is considered ***forgery*** *if* $\Delta \leftarrow \mathsf{SAV.Verify}(\mathsf{pr}, \mathsf{pk}, m^*, \sigma^*, \rho^*, \hat{\tau})$ *is such that* $\Delta \neq \perp$ *and* $\Delta \neq \mathsf{SAV.Verify}(\mathsf{pr}, \mathsf{pk}, m^*, \sigma^*, \rho, \hat{\tau})$*, where* $\rho \leftarrow \mathsf{SAV.Comp}(\mathsf{pb}, \hat{\omega})$ *is generated honestly. A SAV signature scheme is (ε, q_v)-sound against adaptive chosen verification attacks under collusion if* $\mathsf{Prob}\left[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{ACVAuC}}[\lambda] = 1\right] < \varepsilon$*.*

Definition 12 highlights connections between the notions of extended existential unforgeability and soundness against collusion. In particular, it is possible to think of collusion attacks as unforgeability attacks where $\mathcal{A}$ possesses the signing secret key sk (and thus no *sign* query is needed), and a forgery is a tuple for which the output of the server-aided verification does not coincide with the correct one, *e.g.*, if $\sigma^* \leftarrow \mathsf{SAV.Sign}(\mathsf{sk}, m^*)$ then $\mathsf{SAV.Verify}(\mathsf{pr}, \mathsf{pk}, m^*, \sigma^*, \rho^*, \hat{\tau})$ returns 0.

4.3 Anonymity

We initiate the study of anonymity in the context of server-aided verification of signatures and provide the first definition of SAV-anonymity.

Consider the running setting of signed auctions. If a malicious server can distinguish whose signature it is performing the aided-verification of, it can easily 'keep out' target bidders from the auction by preventing their signatures from verifying (in the server aided sense). To prevent such an attack, bidders may want to hide their identity from the untrusted server. SAV-anonymity guarantees precisely this: the auctioneer (trusted verifier) learns the identities of the bidders (signers), while the untrusted server is not able to determine whose signature was involved in the SAV.

Definition 13 (SAV-anonymity). *A SAV scheme is* (ε, q_v)-SAV-*anonymous if* $\mathsf{Prob}\left[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{SAV\text{-}anon}}[\lambda] = 1\right] < \frac{1}{2} + \varepsilon$ *and* $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{SAV\text{-}anon}}[\lambda]$ *is:*

Setup. *The challenger runs the algorithms* $\mathsf{SAV.Init}$, $\mathsf{SAV.VSetup}$ *to obtain the system parameters and the verification keys* $(\mathsf{pb}, \mathsf{pr})$. *Then it runs* $\mathsf{SAV.KeyGen}$ *twice to generate* $(\mathsf{sk}_0, \mathsf{pk}_0), (\mathsf{sk}_1, \mathsf{pk}_1)$ *and draws* $b \xleftarrow{R} \{0,1\}$. $\mathcal{C}$ *gives* pb, $\mathsf{pk}_0, \mathsf{pk}_1$ *to* $\mathcal{A}$ *and retains the secret values* pr, $\mathsf{sk}_0, \mathsf{sk}_1$.

Query I. $\mathcal{A}$ *can adaptively perform up to* q_v *partial-verification queries as follows. The adversary sends a pair* (m, i), $i \in \{0,1\}$ *to* $\mathcal{C}$. *The challenger computes* $\sigma \leftarrow \mathsf{SAV.Sign}(\mathsf{sk}_i, m)$, *runs* $\mathsf{SAV.ProbGen}(\mathsf{pr}, \mathsf{pk}_i, m, \sigma) \rightarrow (\omega, \tau)$ *and returns* ω *to* $\mathcal{A}$.

Challenge. *The adversary chooses a message* m^* *to be challenged on, and sends it to* $\mathcal{C}$. *The challenger computes* $\sigma \leftarrow \mathsf{SAV.Sign}(\mathsf{sk}_b, m^*)$ *and* $(\omega, \tau) \leftarrow \mathsf{SAV.ProbGen}(\mathsf{pr}, \mathsf{pk}_b, m^*, \sigma)$*; and sends* ω *to the adversary.*

Query II. $\mathcal{A}$ *can perform another query phase, as in Query I.*

Output. *The adversary outputs a guess* $b^* \in \{0,1\}$ *for the identity* b *chosen by* $\mathcal{C}$. *The experiment outputs 1 if* $b^* = b$ *and 0 otherwise.*

The fundamental difference between anonymity for signatures schemes [13,27] and SAV-anonymity lies in the choice of the challenge message m^*. In the former case, it is chosen by the challenger at random, while in SAV we let the adversary select it. This change increases the adversary's power and reflects several application scenarios where $\mathcal{A}$ learns the messages (*e.g.*, bids in signed auctions). We remark that in SAV-anonymity the adversary does not have access to the verification outcome Δ, as this would correspond to having a verification oracle, which is not allowed in the anonymity game for signature schemes [13,27].

5 A compiler for SAV

We present here the first generic compiler for server-aided verification of signatures. Our generic composition method allows to combine any signature scheme Σ with an efficient verifiable computation scheme Γ for a function f involved in the signature verification algorithm, and outputs $\mathsf{SAV}^{\Gamma}_{\Sigma}$, a single-round server-aided verification scheme for Σ.

The idea to employ verifiable computation in SAV comes from the following observation. All the attacks presented in [8] succeed because in the target SAV schemes the verifier never checks the validity of the values returned by the server. We leverage the efficiency and security properties of verifiable computation to mitigate such attacks.

5.1 Description of our compiler

Let $\Sigma = (\mathsf{SetUp}_{\Sigma}, \mathsf{KeyGen}_{\Sigma}, \mathsf{Sign}_{\Sigma}, \mathsf{Verify}_{\Sigma})$ be a signature scheme and $\Gamma = (\mathsf{KeyGen}^{\Gamma}, \mathsf{ProbGen}^{\Gamma}, \mathsf{Comp}^{\Gamma}, \mathsf{Verify}^{\Gamma})$ be a verifiable computation scheme.[4] In our generic composition, we identify a computationally-expensive sub-routine of Verify_{Σ} that we refer to as $\mathsf{Ver}_{\mathsf{H}}$ (the *heavy* part of the signature verification); and we outsource $f = \mathsf{Ver}_{\mathsf{H}}$ using the verifiable computation scheme Γ. To ease the presentation, we introduce:

$\mathsf{ProbGen}^{\mathsf{PRE}}$**:** This algorithm prepares the input to $\mathsf{ProbGen}^{\Gamma}$.

$\mathsf{Ver}_{\mathsf{L}}$**:** This algorithm is the computationally *light* part of the signature verification. More precisely, $\mathsf{Ver}_{\mathsf{L}}$ is Verify_{Σ} where $\mathsf{Ver}_{\mathsf{H}}$ is replaced by the output y of Verify^{Γ}. It satisfies: $cost(\mathsf{Ver}_{\mathsf{L}}) < cost(\mathsf{Verify}_{\Sigma})$ and $\mathsf{Ver}_{\mathsf{L}}(\mathsf{pk}_{\Sigma}, m, \sigma, y) = \mathsf{Verify}_{\Sigma}(\mathsf{pk}, m, \sigma)$ whenever $y \neq \perp$.

Definition 14 ($\mathsf{SAV}^{\Gamma}_{\Sigma}$). *Let Σ, Γ and f be as above. Our generic composition method for single-round server-aided verification signature scheme $\mathsf{SAV}^{\Gamma}_{\Sigma}$ is defined by the following possibly randomized algorithms:*

$\mathsf{SAV.Init}(1^{\lambda})$*: the initialisation algorithm outputs the global parameters* $\mathsf{gp} \leftarrow \mathsf{SetUp}_{\Sigma}(1^{\lambda})$*, which are implicitly input to all the algorithms.*

$\mathsf{SAV.KeyGen}()$*: this algorithm outputs* $(\mathsf{pk}_{\Sigma}, \mathsf{sk}_{\Sigma}) \leftarrow \mathsf{KeyGen}_{\Sigma}()$.

$\mathsf{SAV.Sign}(\mathsf{sk}_{\Sigma}, m)$*: the sign algorithm outputs* $\sigma \leftarrow \mathsf{Sign}_{\Sigma}(\mathsf{sk}_{\Sigma}, m)$.

$\mathsf{SAV.VSetup}()$*: the verification setup algorithm outputs a pair of verification keys* $(\mathsf{pk}^{\Gamma}, \mathsf{sk}^{\Gamma}) \leftarrow \mathsf{KeyGen}^{\Gamma}(\lambda, f)$*, where the function f is described in* gp.

$\mathsf{SAV.ProbGen}(\mathsf{sk}^{\Gamma}, \mathsf{pk}_{\Sigma}, m, \sigma)$*: this algorithm first runs* $\mathsf{ProbGen}^{\mathsf{PRE}}(\mathsf{pk}_{\Sigma}, m, \sigma) \rightarrow x$ *to produce an encoding of* pk_{Σ}*,* m*,* σ*. Then x is used to compute the output* $(\omega, \tau) \leftarrow \mathsf{ProbGen}^{\Gamma}(\mathsf{sk}^{\Gamma}, x)$.

$\mathsf{SAV.Comp}(\mathsf{pk}^{\Gamma}, \omega)$*: this algorithm returns* $\rho \leftarrow \mathsf{Comp}^{\Gamma}(\mathsf{pk}^{\Gamma}, \omega)$.

$\mathsf{SAV.Verify}(\mathsf{sk}^{\Gamma}, \mathsf{pk}_{\Sigma}, m, \sigma, \rho, \tau)$*: the verification algorithm executes* $\mathsf{Verify}^{\Gamma}(\mathsf{sk}^{\Gamma}, \rho, \tau) \rightarrow y$*; if $y = \perp$, it sets $\Delta = \perp$ and returns. Otherwise, it returns the output of* $\mathsf{Ver}_{\mathsf{L}}(\mathsf{pk}_{\Sigma}, m, \sigma, y) \rightarrow \Delta \in \{0, 1\}$.

[4] To improve readability, we put the subscript Σ (*resp.* superscript Γ) to each algorithm related to the signature (*resp.* verifiable computation) scheme.

Intuitively, the $\mathsf{SAV.ProbGen}$ algorithm prepares the inputs for the delegated computations (ω) and the private values for the verification of computations (τ). The $\mathsf{SAV.Comp}$ algorithm performs the verifiable delegation of the bilinear pairing computation, and returns ρ, which includes the encoding of the bilinear pairing and some additional values to prove the correctness of the performed operations. Finally, $\mathsf{SAV.Verify}$ checks the correctness of the values received by the server, and proceed with the (light-weight) verification of the signature, only if the server has behaved according to the protocol.

Completeness of $\mathsf{SAV}^{\Gamma}_{\Sigma}$. The correctness of $\mathsf{SAV}^{\Gamma}_{\Sigma}$ is a straight-forward computation assuming that Σ is complete and Γ is correct (see the extended version of this paper [19] for a detailed proof).

Efficiency of $\mathsf{SAV}^{\Gamma}_{\Sigma}$. It is immediate to check that $cost(\mathsf{Verify}_{\Sigma}) = cost(\mathsf{Ver}_{\mathsf{L}}) + cost(\mathsf{Ver}_{\mathsf{H}})$. The $\mathsf{ProbGen}^{\mathsf{PRE}}$ algorithm is just performing encodings of its inputs (usually projections), and does not involve computationally expensive operations.[5] By the efficiency of verifiable computation schemes we have: $cost(\mathsf{Ver}_{\mathsf{H}}) > cost(\mathsf{ProbGen}^{\Gamma}) + cost(\mathsf{Verify}^{\Gamma})$ and thus $cost(\mathsf{Verify}_{\Sigma}) > cost(\mathsf{ProbGen}^{\mathsf{PRE}}) + cost(\mathsf{ProbGen}^{\Gamma}) + cost(\mathsf{Verify}^{\Gamma}) + cost(\mathsf{Ver}_{\mathsf{L}})$, which proves the last claim.

Our generic composition enjoys two additional features: it applies to *any* signature scheme and it allows to reduce the security of $\mathsf{SAV}^{\Gamma}_{\Sigma}$ to the security of its building blocks, Σ and Γ. To demonstrate the first claim, let us set $f = \mathsf{Ver}_{\mathsf{H}} = \mathsf{Verify}_{\Sigma}$ and $\mathsf{ProbGen}^{\mathsf{PRE}}(\mathsf{pk}_{\Sigma}, m, \sigma) \rightarrow x = (\mathsf{pk}_{\Sigma}, m, \sigma)$. The correctness of Γ implies that $y = \mathsf{Ver}_{\mathsf{H}}(x) = \mathsf{Verify}_{\Sigma}(\mathsf{pk}_{\Sigma}, m, \sigma)$. In this case, $\mathsf{Ver}_{\mathsf{L}}(\mathsf{pk}_{\Sigma}, m, \sigma, y)$ is the function that returns 1 if $y = 1$ and 0 otherwise. We defer the proof of the second claim to the following section.

5.2 Security of our generic composition

The following theorems state the security of the compiler presented in Definition 14. Our approach is to identify sufficient requirements on Σ and Γ to guarantee specific security properties in the resulting $\mathsf{SAV}^{\Gamma}_{\Sigma}$ scheme. All the proofs can be found in the extended version of this paper [19]. We highlight that the results below apply to all our instantiations of the SAV signature schemes presented in Section 6, since these are obtained via our generic composition method.

Theorem 1 (Extended Unforgeability of $\mathsf{SAV}^{\Gamma}_{\Sigma}$). *Let Σ be an $(\varepsilon_{\Sigma}, q_s)$-existentially (resp. strongly) unforgeable signature scheme, and Γ an $(\varepsilon^{\Gamma}, q_v)$-secure verifiable computation scheme. Then $\mathsf{SAV}^{\Gamma}_{\Sigma}$ is $(\frac{\varepsilon_{\Sigma}+\varepsilon^{\Gamma}}{2}, q_s, q_v)$-extended existentially (resp. strongly) unforgeable.*

The proof proceeds by reduction transforming type-1a (*resp.* type-1b) forgeries into existential (*resp.* strong) forgeries against Σ; and type-2 forgeries, into forgeries against the security of Γ.

[5] This claim will become clear after seeing examples of SAV signature schemes.

Theorem 2 (Soundness Against Collusion of $\mathsf{SAV}^{\Gamma}_{\Sigma}$). *Let Σ be a correct signature scheme and Γ an $(\varepsilon^{\Gamma}, q_v)$-secure verifiable computation scheme. Then $\mathsf{SAV}^{\Gamma}_{\Sigma}$ is $(\varepsilon^{\Gamma}, q_v)$-sound against collusion.*

The intuition behind the proof of Theorem 2 is the same as in Theorem 1 for the case of type-2 forgeries.

We present now two independent ways to achieve SAV-anonymity for schemes obtained with our compiler: leveraging either the privacy of the verifiable computation scheme or the adaptibility of the signature scheme.

Theorem 3 (Anonymity of $\mathsf{SAV}^{\Gamma}_{\Sigma}$ from Private Verification). *Let Σ be a correct signature scheme and Γ an $(\varepsilon^{\Gamma}, q_v)$-private verifiable computation scheme. Then $\mathsf{SAV}^{\Gamma}_{\Sigma}$ is $(\varepsilon^{\Gamma}, q_v)$-SAV-anonymous.*

Theorem 3 does not require Σ to be anonymous and SAV-anonymity comes directly from the privacy of the verifiable computation scheme.

Key-homomorphic signatures have been recently introduced by Derler and Slamanig [10]. In a nutshell, a signature scheme provides *adaptibility* of signatures [10] if given a signature σ for a message m under a public key pk, it is possible to publicly create a valid σ' for the same message m under a new public key pk'. In particular, there exists an algorithm Adapt that, given pk, m, σ and a shift amount h, returns a pair (pk', σ') for which $\mathsf{Verify}(\mathsf{pk}', m, \sigma') = 1$ (cf. Definition 16 in [10] for a formal statement). [6]

Theorem 4 (Anonymity of $\mathsf{SAV}^{\Gamma}_{\Sigma}$ from Perfect Adaption). *Let Σ be a signature scheme with perfect adaption and Γ a correct verifiable computation scheme. If the output of $\mathsf{ProbGen}^{\mathsf{PRE}}$ depends only on the adapted values, i.e., for all $\mathsf{pr}, \mathsf{pk}, m, \sigma$ there is a function G such that:*

$$\mathsf{ProbGen}^{\mathsf{PRE}}(\mathsf{pr}, \mathsf{pk}, m, \sigma) = \mathsf{G}(\mathsf{Adapt}(\mathsf{pk}, m, \sigma, \mathsf{h}), m)$$

for a randomly chosen shift amount h, then $\mathsf{SAV}^{\Gamma}_{\Sigma}$ is unconditionally SAV-anonymous.

Theorem 4 provides a new application of key-homomorphic signatures to anonymity. The proof is inspired to the tricks used in [10], intuitively SAV-anonymity follows from the indistinguishability of the output of Adapt from $(\mathsf{pk}'', \sigma'' \leftarrow \mathsf{Sign}(\mathsf{sk}'', m))$ for a freshly generated key pair $(\mathsf{pk}'', \mathsf{sk}'')$. Many signatures based on the discrete logarithm problem enjoy this property, *e.g.*, BLS [3] and Wat [24].

6 New instantiations of SAV schemes

Our generic composition requires the existence of a verifiable computation scheme for a function $f = \mathsf{Ver}_{\mathsf{H}}$ used in the signature verification algorithm. To the best

[6] To provide an example, consider the BLS signature scheme [3]. Given $\mathsf{pk} = g^{\mathsf{sk}}$, $m \in \{0,1\}^*$, $\sigma \in \mathbb{G}_p$ and $\mathsf{h} \in \mathbb{Z}_p$, the output of Adapt can be defined as: $\mathsf{pk}' = \mathsf{pk} \cdot g^{\mathsf{h}}$ and $\sigma' = \sigma \cdot H(m)^{\mathsf{h}}$. It is immediate to check that (σ', m) is a valid pair under pk'.

of the authors' knowledge, there are verifiable computation schemes for arithmetic circuits [9,20] and bilinear pairings [6], but no result is yet known for simpler computations such as hash functions and group exponentiations. Following previous works' approach, we consider only SAV for pairing-based signatures [8,21,26,28], since bilinear pairings are bottle-neck computations for resource-limited devices.[7]

All our instantiations of SAV schemes are obtained using the compiler in Definition 14. Their security therefore follows from the results of Section 5.2, once shown that that the chosen schemes satisfy the hypothesis of the theorems. For conciseness, we only define the two algorithms $\mathsf{ProbGen}^{\mathsf{PRE}}$ and $\mathsf{Ver}_{\mathsf{L}}$. Appendix A contains thorough descriptions.

6.1 A secure SAV for BLS ($\mathsf{SAV}^{\mathsf{CDS}_1}_{\mathsf{BLS}}$)

The BLS signature by Boneh *et al.* [3] has been widely used for constructing server-aided verification schemes, *e.g.*, Protocols I and II in [26]. Cao *et al.* [7] and Chow *et al.* [8] have shown that all the existing SAV for BLS are neither existentially unforgeable, nor sound against collusion. This motivates us to propose $\mathsf{SAV}^{\mathsf{CDS}_1}_{\mathsf{BLS}}$ (described in Figure 1). As a verifiable scheme for the pairing computation, we employ *'a protocol for public variable A and B'* by Canard *et al.* [6], which we refer to as CDS_1.

$\mathsf{ProbGen}^{\mathsf{PRE}}(\mathsf{pk}_\Sigma, m, \sigma)$: on input $\mathsf{pk}_\Sigma \in \mathbb{G}_1$, $m \in \{0,1\}^*$ and $\sigma \in \mathbb{G}_1$, the algorithm returns $x = \big((\mathsf{pk}_\Sigma, H(m)), (\sigma, g)\big)$.

$\mathsf{Ver}_{\mathsf{L}}(\mathsf{pk}_\Sigma, m, \sigma, y)$: this algorithm is $\mathsf{Verify}_{\mathsf{BLS}}$ where the computation of the two pairings is replaced with the output $y = (y_1, y_2)$ of $\mathsf{Verify}^{\mathsf{CDS}_2}$. Formally, $\mathsf{Ver}_{\mathsf{L}}$ checks whether $y_1 = y_2$, in which case it outputs $\Delta = 1$, otherwise it returns $\Delta = 0$.

Fig. 1. The core algorithms of $\mathsf{SAV}^{\mathsf{CDS}_1}_{\mathsf{BLS}}$.

By the correctness of the CDS_1 scheme $y_2 = e(\mathsf{pk}_\Sigma, H(m))$ and $y_2 = e(\sigma, g)$, thus $\mathsf{Ver}_{\mathsf{L}}$ has the same output as $\mathsf{Verify}_{\mathsf{BLS}}$. Given that BLS is strongly unforgeable in the random oracle model [3] and that CDS_1 is secure in the generic group model [6], $\mathsf{SAV}^{\mathsf{CDS}_1}_{\mathsf{BLS}}$ is extended strongly unforgeable and sound against collusion. Our SAV scheme for the BLS is not SAV-anonymous: the signer's public key is given to the server for the aided verification. However, SAV-anonymity can be simply gained via the adaptability of BLS [10].

In $\mathsf{SAV}^{\mathsf{CDS}_1}_{\mathsf{BLS}}$ the verifier does not need to perform *any* pairing computation. This is a very essential feature, especially if the verifying device has very limited computational power, *e.g.*, an RFID tag.

[7] To give benchmarks, let M_p denote the computational cost of a base field multiplication in $\mathbb{F}_p$ with $\log p = 256$, then computing z^a for any $z \in \mathbb{F}_p$ and $a \in [p]$ costs about $256M_p$, while computing the Optimal Ate pairing on the BN curve requires about $16000M_p$ (results extrapolated from Table 1 in [16]).

6.2 A secure SAV for Wat ($\mathsf{SAV}^{\mathsf{CDS_1}}_{\mathsf{Wat}}$)

Wu *et al.* [26] proposed a SAV for Waters' signature Wat [24], which is neither existentially unforgeable nor sound against collusion. Here we propose $\mathsf{SAV}^{\mathsf{CDS_1}}_{\mathsf{Wat}}$ (described in Figure 2), which is similar to Protocol III in [26], but has strong security guarantees thanks to the verifiable computation scheme for *'public A and B'* $\mathsf{CDS_1}$ [6].

$\mathsf{ProbGen}^{\mathsf{PRE}}(\mathsf{pk}_\Sigma, m, \sigma)$: given $\mathsf{pk}_\Sigma \in \mathbb{G}_1$, $m \in \{0,1\}^*$ and $\sigma \in \mathbb{G}_1$, select $\mathsf{h} \xleftarrow{R} \mathbb{Z}_p$, compute $(\mathsf{pk'}_\Sigma, \sigma') \leftarrow \mathsf{Adapt}(\mathsf{pk}_\Sigma, m, \sigma, \mathsf{h})$, return $x = (\mathsf{pk'}_\Sigma, m, \sigma')$.

$\mathsf{Ver_L}(\mathsf{pk'}_\Sigma, m, \sigma, y)$: this is $\mathsf{Verify_{Wat}}$ where the computation of the two pairings is replaced with the outputs y_1, y_2 of $\mathsf{Verify}^{\mathsf{CDS_1}}$. Formally, $\mathsf{Ver_L}$ checks if $y_1 = \mathsf{pk'}_\Sigma \cdot y_2$, in which case it outputs $\Delta = 1$, otherwise it returns $\Delta = 0$.

Fig. 2. The core algorithms of $\mathsf{SAV}^{\mathsf{CDS_1}}_{\mathsf{Wat}}$.

By the correctness of the $\mathsf{CDS_1}$ scheme $y_1 = e(\sigma_1, g)$, and $y_2 = e(H(m), \sigma_2)$. Thus, $\mathsf{Ver_L}$ has the same output as $\mathsf{Verify_{Wat}}$. Given that $\mathsf{CDS_1}$ is secure in the generic group model [6], and that Wat is existentially unforgeable in the standard model [24] our $\mathsf{SAV}^{\mathsf{CDS_1}}_{\mathsf{Wat}}$ is extended existential unforgeable and sound against collusion. Similarly to Protocol III in [26], $\mathsf{SAV}^{\mathsf{CDS_1}}_{\mathsf{Wat}}$ achieves SAV-anonymity thanks to the perfect adaption of Wat [10].

6.3 The first SAV for CL ($\mathsf{SAV}^{\mathsf{CDS_2}}_{\mathsf{CL}}$)

The verification of the BLS and the Wat signatures only requires the computation of two bilinear pairings. We want to move the focus to more complex signature schemes that would benefit more of server-aided verification. To this end, we consider scheme A by Camenish and Lysyanskaya [5], which we refer to as CL, where $\mathsf{Verify_{CL}}$ involves the computation of five bilinear pairings. For verifiability we employ $\mathsf{CDS_2}$, *'a protocol with public constant B and variable secret A'* by Canard *et al.* [6]. Our $\mathsf{SAV}^{\mathsf{CDS_2}}_{\mathsf{CL}}$ scheme is reported in Figure 3.

$\mathsf{ProbGen}^{\mathsf{PRE}}(\mathsf{pk}_\Sigma, m, \sigma)$: this algorithm simply returns the first two entries of the signature $\sigma = (\sigma_1, \sigma_2, \sigma_3)$, *i.e.*, $x = (\sigma_1, \sigma_2)$.

$\mathsf{Ver_L}(\mathsf{pk}_\Sigma, m, \sigma, y)$: this algorithm is $\mathsf{Verify_{CL}}$, except for two pairing computations which are replaced with the outcome $y = (\beta_1, \beta_2)$ of $\mathsf{Verify}^{\mathsf{CDS_2}}$. More precisely, the $\mathsf{Ver_L}$ algorithm computes $\alpha_1 = e(\sigma_1, Y)$, $\alpha_2 = e(X, \sigma_1)$ and $\alpha_3 = e(X, \sigma_2)^m$. It then checks whether $\alpha_1 = \beta_1$ and $\alpha_2 \cdot \alpha_3 = \beta_2$. If both of the conditions hold, the algorithm returns $\Delta = 1$, otherwise $\Delta = 0$.

Fig. 3. The core algorithms of $\mathsf{SAV}^{\mathsf{CDS_2}}_{\mathsf{CL}}$.

By the correctness of $\mathsf{CDS_2}$ we have: $y_1 = \beta_1 = e(\sigma, g)$, and $y_2 = \beta_2 = e(H(m), \mathsf{pk}_\Sigma)$. Therefore $\mathsf{Ver_L}$ performs the same checks as $\mathsf{Verify_{CL}}$ and the two algorithms have the same output. Given that CL is existential unforgeable in the standard model [5] and $\mathsf{CDS_2}$ is secure and private in the generic group model [6], $\mathsf{SAV}^{\mathsf{CDS_2}}_{\mathsf{CL}}$ is extended-existential unforgeable, sound against collusion

and SAV-anonymous. Therefore $\mathsf{SAV}^{\mathsf{CDS}_2}_{\mathsf{CL}}$ is an example of a scheme which is SAV-anonymous although the base signature scheme is not anonymous.

6.4 Comparison with previous work

Table 1 gives a compact overview of how our SAV schemes compare to previous proposals in terms of unforgeability, soundness under collusion and SAV-anonymity. We report only the highest level of unforgeability that the scheme provides. A yes (*resp.* no) in the table states that the scheme does (*resp.* does not) achieve the property written at the beginning of the row, *e.g.*, Protocol III does not employ a verifiable computation scheme and provides SAV-anonymity. Every scheme or property is followed by a reference paper or the section where the claim is proven.

	Protocol I [26]	Protocol II [26]	$\mathsf{SAV}^{\mathsf{CDS}_1}_{\mathsf{BLS}}$	Protocol III [26]	$\mathsf{SAV}^{\mathsf{CDS}_1}_{\mathsf{Wat}}$	SAV-ZSS [15]	$\mathsf{SAV}^{\mathsf{CDS}_2}_{\mathsf{CL}}$
signature	BLS [3]	BLS [3]	BLS [3]	Wat [24]	Wat [24]	ZSS [28]	CL [5]
verifiability	no	no	CDS_1 [6]	no	CDS_1 [6]	no	CDS_2 [6]
unforgeability	EUF [26]	no [8]	ExSUF (6.1)	no [19]	ExEUF (6.2)	EUF [15]	ExEUF (6.3)
collusion resistance	no [8]	no [19]	yes (6.1)	no [19]	yes (6.2)	no [19]	yes (6.3)
anonymity	no [19]	no [19]	no (6.1)	yes [19]	yes (6.2)	no [19]	yes (6.3)

Table 1. Comparison among our SAV schemes and previous works: Protocol I (Figure 3 in [26]), Protocol II (Figure 5 in [26]), Protocol III (Figure 4 in [26]), SAV-ZSS [15] (depicted in Figure 1 in [26]).

Regarding efficiency, the computational cost of pairing-based algorithms is influenced by three main parameters: (*i*) the elliptic curve, (*ii*) the field size, and (*iii*) the bilinear pairing. As a result, it is impossible to state that a given algorithm is efficient for all pairings and for all curves, since even the computational cost of the most basic operations (*e.g.*, point addition) variates significantly with the above parameters. For example, CDS_2 provides a 70% efficiency gain[8] for the delegator (verifier) when the employed pairing is the Optimal Ate pairing on the KSS-18 curve [6], but is nearly inefficient when computed on the BN curve [16].

7 Conclusions

In this paper, we provided a framework for single-round server-aided verification signature schemes and introduced a security model which extends previous proposals towards more realistic attack scenarios and stronger adversaries. In addition, we defined the first generic composition method to obtain a SAV for any signature scheme using an appropriate verifiable computation scheme. Our compiler identifies for the first time sufficient requirements on the underlying primitives to ensure the security and anonymity of the resulting SAV scheme.

[8] Efficiency gain is the ratio $\big(cost(\mathsf{SAV.ProbGen}) + cost(\mathsf{SAV.Verify})\big)/cost(\mathsf{Verify}_{\Sigma})$.

In particular, we showed sufficient conditions to achieve both computational and unconditional SAV-anonymity. Finally, we introduced three new SAV signature schemes obtained via our generic composition method, that simultaneously achieve existential unforgeability and soundness against collusion.

Currently, Canard *et al.*'s is the only verifiable computation scheme for pairings available in the literature. Considering the wide applicability of bilinear pairings in cryptography, a more efficient verifiable computation scheme for these functions would render pairings a server-aided accessible computation to a large variety of resource-limited devices, such as the ones involved in IoT and cloud computing settings.

Acknowledgements We thank Dario Fiore (Assistant Research Professor) for providing useful comments on the contributions of this paper. This work was partially supported by the Japanese Society for the Promotion of Science (JSPS), summer program, the SNSF project SwissSenseSynergy and the STINT project IB 2015-6001.

References

1. P. Béguin and J. Quisquater. Fast server-aided RSA signatures secure against active attacks. In *Advances in Cryptology - CRYPTO '95*, pages 57–69, 1995.
2. S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *Annual Cryptology Conference*, pages 111–131. Springer, 2011.
3. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.
4. D. Boneh, E. Shen, and B. Waters. Strongly unforgeable signatures based on computational diffie-hellman. In *International Workshop on Public Key Cryptography*, pages 229–240. Springer, 2006.
5. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Annual International Cryptology Conference*, pages 56–72. Springer, 2004.
6. S. Canard, J. Devigne, and O. Sanders. Delegating a pairing can be both secure and efficient. In *International Conference on Applied Cryptography and Network Security*, pages 549–565. Springer, 2014.
7. Z. Cao, L. Liu, and O. Markowitch. On two kinds of flaws in some server-aided verification schemes. *International Journal of Network Security*, 18(6):1054–1059, 2016.
8. S. S. Chow, M. H. Au, and W. Susilo. Server-aided signatures verification secure against collusion attack. *Inf. Security Tech. Report*, 17(3):46 – 57, 2013.
9. C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur. Geppetto: Versatile verifiable computation. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 253–270. IEEE, 2015.
10. D. Derler and D. Slamanig. Key-homomorphic signatures and applications to multiparty signatures. Technical report, IACR Cryptology ePrint Archive 2016, 792, 2016.
11. X. Ding, D. Mazzocchi, and G. Tsudik. Experimenting with server-aided signatures. In *NDSS*, 2002.

12. D. Fiore, R. Gennaro, and V. Pastro. Efficiently verifiable computation on encrypted data. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 844–855. ACM, 2014.
13. M. Fischlin. Anonymous signatures made easy. In *International Workshop on Public Key Cryptography*, pages 31–42. Springer, 2007.
14. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Annual Cryptology Conference*, pages 465–482. Springer, 2010.
15. M. Girault and D. Lefranc. Server-aided verification: theory and practice. In *International Conference on the Theory and Application of Cryptology and Information Security - ASIACRYPT*, pages 605–623. Springer, 2005.
16. A. Guillevic and D. Vergnaud. Algorithms for outsourcing pairing computation. In *CARDIS*, pages 193–211. Springer, 2014.
17. F. Guo, Y. Mu, W. Susilo, and V. Varadharajan. Server-aided signature verification for lightweight devices. *The Computer Journal*, page bxt003, 2013.
18. C. H. Lim and P. J. Lee. Server (prover/signer)-aided verification of identity proofs and signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques - EUROCRYPT*, pages 64–78. Springer, 1995.
19. E. Pagnin, A. Mitrokotsa, and K. Tanaka. Anonymous single-round server-aided verification. 2017. `http://eprint.iacr.org/2017/794`.
20. B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 238–252. IEEE, 2013.
21. B. Wang. A server-aided verification signature scheme without random oracles. *International Review on Computers & Software*, 7:3446, 2012.
22. Z. Wang. A new construction of the server-aided verification signature scheme. *Mathematical and Computer Modelling*, 55(1):97–101, 2012.
23. Z. Wang, L. Wang, Y. Yang, and Z. Hu. Comment on wu et al.'s server-aided verification signature schemes. *IJ Network Security*, 10(2):158–160, 2010.
24. B. Waters. Efficient identity-based encryption without random oracles. *In EUROCRYPT 2005, in: Lecture Notes in Computer Science*, 3494(114–127), 2005.
25. W. Wu, Y. Mu, W. Susilo, and X. Huang. Server-aided verification signatures: Definitions and new constructions. In *International Conference on Provable Security*, pages 141–155. Springer, 2008.
26. W. Wu, Y. Mu, W. Susilo, and X. Huang. Provably secure server-aided verification signatures. *Computers & Mathematics with Applications*, 61(7):1705 – 1723, 2011.
27. G. Yang, D. S. Wong, X. Deng, and H. Wang. Anonymous signature schemes. In *International Workshop on Public Key Cryptography*, pages 347–363. Springer, 2006.
28. F. Zhang, R. Safavi-Naini, and W. Susilo. An efficient signature scheme from bilinear pairings and its applications. In *International Workshop on Public Key Cryptography*, pages 277–290. Springer, 2004.

A Detailed descriptions of our SAV schemes

In this Appendix we present thorough descriptions of the new SAV scheme proposed in this paper (Section 6). The complete explanations of the algorithms in $\mathsf{SAV}^{\mathsf{CDS}_1}_{\mathsf{BLS}}$, $\mathsf{SAV}^{\mathsf{CDS}_1}_{\mathsf{Wat}}$ and $\mathsf{SAV}^{\mathsf{CDS}_2}_{\mathsf{CL}}$ are presented in Figures 4, 5 and 6 respectively.

For consistency, we adopt the multiplicative notation for describing the operation elliptic curve groups.

$\mathsf{SAV.Init}(1^\lambda) = \mathsf{SetUp}_{\mathsf{BLS}}(1^\lambda)$. This algorithm generates the global parameters of the scheme, that include: a Gap Diffie-Hellman bilinear group $(p, g, \mathbb{G}, \mathbb{G}_T, e)$ according to the security parameter λ; and a hash function $H : \{0,1\}^* \to \mathbb{G}$ that maps messages $m \in \mathcal{M} = \{0,1\}^*$ to group elements in $\mathbb{G}$. The output is $\mathsf{gp} = (p, g, H, \mathbb{G}, \mathbb{G}_T, e)$.

$\mathsf{SAV.KeyGen}() = \mathsf{KeyGen}_{\mathsf{BLS}}()$. The key generation algorithm draws a random $s \xleftarrow{R} \mathbb{Z}_p^*$ and outputs $(\mathsf{pk}, \mathsf{sk}) = (g^s, s)$.

$\mathsf{SAV.VSetup}() = \mathsf{KeyGen}^{\mathsf{CDS}_1}()$. This algorithm outputs $\mathsf{pr} = \mathsf{void}$ and $\mathsf{pb} = (p, \mathbb{G}, \mathbb{G}_T, e, g, \hat{\beta})$, where $\hat{\beta} = e(g, g)$.

$\mathsf{SAV.Sign}(\mathsf{sk}, m) = \mathsf{Sign}_{\mathsf{BLS}}(\mathsf{sk}, m)$. The signing algorithm outputs $\sigma = H(m)^s \in \mathbb{G}$.

$\mathsf{SAV.ProbGen}(\mathsf{void}, \mathsf{pk}, m, \sigma)$. This algorithm runs $\mathsf{ProbGen}^{\mathsf{PRE}}(\mathsf{pk}, m, \sigma) \to \big((\mathsf{pk}, H(m)), (\sigma, g)\big)$ and returns the outputs of $\mathsf{ProbGen}^{\mathsf{CDS}_1}$ on the two pairs $(\mathsf{pk}, H(m))$ and (σ, g). In details, for $(\mathsf{pk}, H(m))$ the problem generator algorithm selects two random values $r_1, r_2 \xleftarrow{R} \mathbb{Z}_p$, computes the points $R_1 = \mathsf{pk}^{r_2^{-1}} g^{r_1}$, $R_2 = H(m)^{r_1^{-1}} g^{r_2}$ and $\hat{U} = \hat{\beta}^{r_1 r_2}$. This process (with fresh randomness) is applied to the pair (σ, g) as well. The final outputs are $\omega = \big((\mathsf{pk}, H(m), R_1^{(1)}, R_2^{(1)}), (\sigma, g, R_1^{(2)}, R_2^{(2)})\big)$ and $\tau = \big((\hat{U}^{(1)}, r_1^{(1)}, r_2^{(1)}), (\hat{U}^{(2)} r_1^{(2)}, r_2^{(2)})\big)$.

$\mathsf{SAV.Comp}(\mathsf{pb}, \omega)$. The algorithm computes the following bilinear pairings: $\alpha_1^{(1)} = e(\mathsf{pk}, H(m))$, $\alpha_2^{(1)} = e(R_1^{(1)}, R_2^{(1)})\big(e(\mathsf{pk}, g) e(g, H(m))\big)^{-1}$, $\alpha_1^{(2)} = e(\sigma, g)$, $\alpha_2^{(2)} = e(R_1^{(2)}, R_2^{(2)})\big(e(\sigma, g) e(g, g)\big)^{-1}$. It returns $\rho = (\rho_1, \rho_2) = \big((\alpha_1^{(1)}, \alpha_2^{(1)}), (\alpha_1^{(2)}, \alpha_2^{(2)})\big)$.

$\mathsf{SAV.Verify}(\mathsf{void}, \mathsf{pk}, m, \sigma, \rho, \tau)$. The verification algorithm first runs $\mathsf{Verify}^{\mathsf{CDS}_1}(\rho_i, \tau_i)$ for $i \in [2]$, *i.e.*, checks whether $\alpha_2^{(i)} = \hat{U}^{(i)} (\alpha_1^{(i)})^{(r_1^{(i)} r_2^{(i)})^{-1}}$ and $\alpha_1 \in \mathbb{G}_T$. If any of the previous checks fails, the verification algorithm returns $\Delta = \perp$ and halts. Otherwise, it sets $y_i = \alpha_1^{(i)}$, for $i \in [2]$ and runs $\mathsf{Ver}_{\mathsf{L}}(\mathsf{pk}, m, \sigma, y)$, which returns $\Delta = 1$ if $y_1 = y_2$, and $\Delta = 0$ otherwise.

Fig. 4. $\mathsf{SAV}^{\mathsf{CDS}_1}_{\mathsf{BLS}}$: Our SAV for the BLS Signature in [3].

$\mathsf{SAV.Init}(1^\lambda) = \mathsf{SetUp}_{\mathsf{Wat}}(1^\lambda)$. This algorithm generates a bilinear group $(p, g, \mathbb{G}, \mathbb{G}_T, e)$ according to the security parameter λ; selects $n+1$ group elements $V_0, V_1, \ldots V_n \xleftarrow{R} \mathbb{G}$ and defines a function $H : \{0,1\}^n \to \mathbb{G}$ as $H(m) = V_0(\prod_{i=1}^{n} V_i^{m_i})$. The output is $\mathsf{gp} = (p, g, H, \mathbb{G}, \mathbb{G}_T, e)$.

$\mathsf{SAV.KeyGen}() = \mathsf{KeyGen}_{\mathsf{Wat}}()$. The key generation algorithm draws a random $s \xleftarrow{R} \mathbb{Z}_p^*$ and outputs $(\mathsf{pk}, \mathsf{sk}) = (e(g,g)^s, s)$.

$\mathsf{SAV.VSetup}() = \mathsf{KeyGen}^{\mathsf{CDS}_1}()$. This algorithm outputs $\mathsf{pr} = \mathsf{void}$ and $\mathsf{pb} = (p, \mathbb{G}, \mathbb{G}_T, e, g, \hat{\beta})$, where $\hat{\beta} = e(g,g)$.

$\mathsf{SAV.Sign}(\mathsf{sk}, m) = \mathsf{Sign}_{\mathsf{Wat}}(\mathsf{sk}, m)$. The signing algorithm picks a random $a \xleftarrow{R} \mathbb{Z}_p$ and outputs $\sigma = (\sigma_1, \sigma_2) = (g^s(H(m))^a, g^a) \in \mathbb{G}^2$.

$\mathsf{SAV.ProbGen}(\mathsf{void}, \mathsf{pk}, m, \sigma)$. This algorithm runs $\mathsf{ProbGen}^{\mathsf{PRE}}(\mathsf{pk}, m, \sigma) \to (\mathsf{pk}', \sigma')$ to create a signature for a new public key, *i.e.*, it picks two random values $\mathsf{h}, b \xleftarrow{R} \mathbb{Z}_p$ and sets $\mathsf{pk}' = \mathsf{pk}\hat{\beta}^{\mathsf{h}}$, $\sigma' = (g^{\mathsf{h}}\sigma_1 H(m)^b, \sigma_2 g^b)$. *(By the adaptivity of* Wat *if σ is a valid signature for m under* sk *with randomness a, then σ' is a valid signature for m under* $\mathsf{sk}' + \mathsf{h}$ *with randomness $a + b$).*
Secondly, the problem generation algorithm runs $\mathsf{ProbGen}^{\mathsf{CDS}_1}$ on (σ_1', g) and $(H(m), \sigma_2')$. In details, for each pair (A, B), the algorithm selects two random values $r_1, r_2 \xleftarrow{R} \mathbb{Z}_p$, computes the points $R_1 = A^{r_2^{-1}} g^{r_1}$, $R_2 = B^{r_1^{-1}} g^{r_2}$ and $\hat{U} = \hat{\beta}^{r_1 r_2}$. The final outputs are $\omega = (R_1^{(1)}, R_2^{(1)}, R_1^{(2)}, R_2^{(2)})$ and $\tau = (\mathsf{pk}'\hat{U}^{(1)}, r_1^{(1)}, r_2^{(1)}, \hat{U}^{(2)} r_1^{(2)}, r_2^{(2)})$.

$\mathsf{SAV.Comp}(\mathsf{pb}, \omega)$. The algorithm parses ω as $\big((R_1^{(1)}, R_2^{(1)}), (R_1^{(2)}, R_2^{(2)})\big)$; for each pair (A, B) it computes $\alpha_1 = e(A, B)$ and $\alpha_2 = e(R_1, R_2)\big(e(g, B), e(A, g)\big)^{-1}$. It returns $\rho = (\alpha_1^{(1)}, \alpha_2^{(1)}, \alpha_1^{(2)}, \alpha_2^{(2)})$.

$\mathsf{SAV.Verify}(\mathsf{void}, \mathsf{pk}, m, \sigma, \rho, \tau)$. The verification algorithm parses $\rho = (\rho^{(1)}, \rho^{(2)}) = \big((\alpha_1^{(1)}, \alpha_2^{(1)}), (\alpha_1^{(2)}, \alpha_2^{(2)})\big)$ and $\tau = (\mathsf{pk}', \tau^{(1)}, \tau^{(2)}) = \big((\hat{U}^{(1)}, r_1^{(1)}, r_2^{(1)}), (\hat{U}^{(2)} r_1^{(2)}, r_2^{(2)})\big)$. For $i \in [2]$ it runs $\mathsf{Verify}^{\mathsf{CDS}_2}(\rho^{(i)}, \tau^{(i)})$, *i.e.*, it checks if $\alpha_2^{(i)} = \hat{U}^{(i)}(\alpha_1^{(i)})^{(r_1^{(i)} r_2^{(i)})^{-1}}$ and $\alpha_1 \in \mathbb{G}_T$. If any of the previous checks fails, the verification algorithm returns $\Delta = \perp$ and halts. Otherwise, it returns $y^{(i)} = \alpha_1^{(i)}$ and runs $\mathsf{Ver}_{\mathsf{L}}(\mathsf{pk}, m, \sigma, y)$, which returns $\Delta = 1$ if $y^{(1)} = \mathsf{pk}'\, y^{(2)}$, and $\Delta = 0$ otherwise.

Fig. 5. $\mathsf{SAV}_{\mathsf{Wat}}^{\mathsf{CDS}_1}$: Our SAV for the Wat Signature in [24].

$\mathsf{SAV.Init}(1^\lambda) = \mathsf{SetUp}_{\mathsf{CL}}(1^\lambda)$. The setup algorithm generates the global parameters of the scheme, that include a bilinear group $(q, \mathbb{G}, g, \mathbb{G}_T, \hat{g}, e)$.

$\mathsf{SAV.KeyGen}() = \mathsf{KeyGen}_{\mathsf{CL}}()$. The key generation algorithm draws two random values $x, y \xleftarrow{R} \mathbb{Z}_q$, computes $g^x = X,\ g^y = Y$ and returns $\mathsf{pk} = (X, Y)$ and $\mathsf{sk} = (x, y)$.

$\mathsf{SAV.VSetup}() = \mathsf{KeyGen}^{\mathsf{CDS}_2}()$. This algorithm outputs $\mathsf{pr} = \mathsf{void}$ and $\mathsf{pb} = (p, \mathbb{G}, \mathbb{G}_T, e, G, B, \hat{\beta})$, where $G \xleftarrow{R} \mathbb{G}$, $B = g$ and $\hat{\beta} = e(G, B)$.

$\mathsf{SAV.Sign}(\mathsf{sk}, m) = \mathsf{Sign}_{\mathsf{CL}}(\mathsf{sk}, m)$. The sign algorithm picks a random $a \xleftarrow{R} \mathbb{G}$ and outputs the signature $\sigma = (\sigma_1, \sigma_2, \sigma_3) = (a, a^y, a^{x+mxy}) \in \mathbb{G}^3$.

$\mathsf{SAV.ProbGen}(\mathsf{void}, \mathsf{pk}, m, \sigma)$. This algorithm first runs $\mathsf{ProbGen}^{\mathsf{PRE}}(\mathsf{pk}, m, \sigma) \to (\sigma_2, \sigma_3)$. Then it runs $\mathsf{ProbGen}^{\mathsf{CDS}_2}$ on σ_2 and σ_3. In more details, for $i \in \{2, 3\}$ it selects three random values $r_1^{(i)}, r_2^{(i)}, u^{(i)} \xleftarrow{R} \mathbb{Z}_q$, computes the points $R_1^{(i)} = \sigma_i \cdot G^{r_1^{(i)}}$ and $R_2^{(i)} = \sigma_i^{u^{(i)}} \cdot G^{r_2^{(i)}}$, and calculates $\hat{X}_1^{(i)} = (\hat{\beta})^{r_1^{(i)}}$, $\hat{X}_2^{(i)} = (\hat{\beta})^{r_2^{(i)}}$. The final outputs are $\omega = (R_1^{(2)}, R_2^{(2)}, R_1^{(3)}, R_2^{(3)})$ and $\tau = (u^{(2)}, \hat{X}_1^{(2)}, \hat{X}_2^{(2)}, u^{(3)}, \hat{X}_1^{(3)}, \hat{X}_2^{(3)})$.

$\mathsf{SAV.Comp}(\mathsf{pb}, \omega)$. The algorithm parses $\omega = (R_1^{(2)}, R_2^{(2)}, R_1^{(3)}, R_2^{(3)})$ and returns $\rho = (e(R_1^{(2)}, g), e(R_2^{(2)}, g), e(R_1^{(3)}, g), e(R_2^{(2)}, g))$.

$\mathsf{SAV.Verify}(\mathsf{void}, \mathsf{pk}, m, \sigma, \rho, \tau)$. The verification algorithm first runs $\mathsf{Verify}^{\mathsf{CDS}_2}(\rho, \tau)$, *i.e.*, for $i \in \{2, 3\}$ it checks if $\alpha_2^{(i)} = \hat{X}_2^{(i)}(\alpha_1(\hat{X}_1^{(i)})^{-1})^u$ and $\alpha_1^{(i)} \in \mathbb{G}_T$. If any of the previous checks fails, the verification algorithm returns $\Delta = \perp$ and halts. Otherwise, the values $y^{(i)} = \beta_1^{(i)}(\hat{X}_1^{(i)})^{-1}$, for $i \in \{2, 3\}$ are used as input for $\mathsf{Ver}_{\mathsf{L}}$. In details, $\mathsf{Ver}_{\mathsf{L}}(\mathsf{pk}, m, \sigma, y = (y^{(2)}, y^{(3)}))$, computes: $\beta_1 = e(\sigma_1, Y)$, $\beta_2 = e(X, \sigma_1\sigma_2^m)$. If both $\beta_1 = y(1)$ and $\beta_2 = y^{(2)}$, the algorithm returns $\Delta = 1$; otherwise it returns $\Delta = 0$.

Fig. 6. $\mathsf{SAV}_{\mathsf{CL}}^{\mathsf{CDS}_2}$: Our SAV for the CL Signature in [5].