Server-Aided Signatures Verification Secure against Collusion Attack

Sherman S. M. Chow
University of Waterloo
Ontario, Canada
smchow@uwaterloo.ca

Man Ho Au
University of Wollongong
Australia
aau@uow.edu.au

Willy Susilo
University of Wollongong
Australia
wsusilo@uow.edu.au

ABSTRACT

Wireless handheld devices which support e-mail and web browsing are increasingly popular. The authenticity of the information received is important, especially for business uses. In server-aided verification (SAV), a substantial part of the verification computation can be offloaded to a powerful but possibly untrusted server. This allows resource-constrained devices to enjoy the security guarantees provided by cryptographic schemes, such as pairing-based signatures, which may be too heavyweight to verify otherwise.

To gain unfair advantage, an adversary may bribe the server to launch various kinds of attacks – to convince that an invalid signature held by a client is a valid one (say for providing false information or repudiable commitment) or to claim that a valid signature is invalid (say for spoiling the offer provided by an opponent). However, these concerns are not properly captured by existing security models.

In this paper, we provide a generic pairing-based SAV protocol. Compared with the protocol of Girault and Lefranc in Asiacrypt '05, ours provides a higher level of security yet applicable to a much wider class of pairing-based cryptosystems. In particular, it suggests SAV protocols for short signatures in the standard model and aggregate signatures which have not been studied before.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security

Keywords

server-aided computation, collusion attack, authenticity, signatures, pairings, delegation, weak computational device

ASIACCS '11, March 22–24, 2011, Hong Kong, China.

Keywords

server-aided computation; pairings; signatures; delegation;

1. INTRODUCTION

Server-aided computation protocols are protocols in which a device, also called the client, offloads the computation to a server. Nowadays, there are many computation devices with relatively limited power, e.g., smart card, RFID tag, but need to perform cryptographic operations which are both time and power consuming. Server-aided computation protocols are particularly useful since these devices often have access to a more powerful device which can act as the server.

1.1 Server-Aided Computation of Pairing

Bilinear pairing is an enabling technology for constructing next-generation cryptosystems. On one hand, it allows the construction of novel cryptographic applications that are otherwise difficult or impossible to build based on other primitives. On the other hand, it also facilitates the construction of some highly efficient cryptographic solutions which are particularly appealing for computationally-limited devices. Pairing is currently instantiated on elliptic curves. The use of elliptic curves leads to a shorter-representation of the cryptographic group elements, so many pairing-based schemes can be made space-efficient. An ever increasing number of protocols have been appearing in the literature since its introduction in constructing cryptographic schemes. Some of which have a very simple design (e.g., [3] and [2]) when compared with non-pairing constructions. While these schemes may offer high efficiency in terms of bandwidth or storage requirement, they often require relatively heavyweight cryptographic operations. To enjoy the features made possible by cryptography, one may resorting to a symmetric-key solution instead of a public-key one (e.g., [7]), or employ tricks such as precomputation (e.g., [5]). In this paper, we study the approach of server-aided computation, with focus in signature verification.

1.2 Collusion Attack

Server-aided computation should not introduce new vulnerability to the application. This is true if the server is fully-trusted and there exists a secure channel between it and the client. In this case the client can simply offload the computation to the server. Unfortunately, this trust assumption is not realistic. In reality, the client could be facing a malicious server, or the client cannot establish an authenticated connection with a trusted server easily. (Authenticated connection can be established by cryptographic

means, which leads to a chicken-and-egg situation). Moreover, concerns with confidentiality arise when the operation involves some private information from the client (e.g., decryption or signing which uses the client's private key) or when the operation result gives the basis for the client to make a decision whether the integrity of some information (e.g., via digital signature) has been tampered or not.

Wireless handheld devices which support push e-mail, text messaging, and web browsing are increasingly popular for business uses. For examples, roaming employees can response quickly to their colleagues, and traders can always get update information of the exchange market. In these scenarios, it is important to verify the authenticity of the messages received. It is also undesirable if a malicious server can falsely claim that a valid signature is an invalid one, which effectively denies the client from receiving legitimate message. This attack may be motivated by a unfair profit gain, say a bid from a competitor of an attacker can be "made" invalid and now this attacker can supply another bid.

1.3 Existing Work

In the work of Girault and Lefranc [9], some security requirements of server-aided verification are formalized. Two existing protocols are then analyzed in their model and a new one for a limited class of pairing-based signature schemes has been introduced (specifically, schemes where the verification requires only comparisons against a fixed element $\hat{e}(g,g)$). In their model, a malicious server is not allowed to collude with the signer. In particular, the security of the protocol relies that the malicious server is not given any valid signature on that message. Subsequently, Wu *et al.* [13] proposed a SAV protocol for Boneh-Lynn-Shacham signature [3] since it is not covered in the previous semi-generic construction. Wu *et al.* also observed that the security model of Girault and Lefranc [9] does not allow an adversary to interact with the client via the aided verification protocol, and consequently devised an attack of their protocol by exploiting this weakness. However, this attack actually assumes the server to be deviated from the specification of the aided verification protocol, which is different from the models considered in [9, 13]. On the other hand, Wu *et al.* [13] proposed a security model aimed to capture the collusion of the signer with the server. Recently, [12] criticized the model of [13] and proposed a new one. We found that these two models may not model the collusion attack well. In particular, they do not address the situation when the attacker is in fact in possession of the private key of the signer. For example, a malicious signer, after created an invalid signature, may collude with the server and try to convince the client to accept this maliciously generated signature.

2. PRELIMINARIES

For a finite set $\mathcal{S}$, $x \in_R \mathcal{S}$ means choosing an element x uniformly at random from $\mathcal{S}$. For algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, we use $out \xleftarrow{\$} \mathcal{A}_1(in)$ to denote that out is the result of running $\mathcal{A}_1$ on input in, and $out_1 \leftarrow \mathcal{A}_1(in_1) \leftrightarrow \mathcal{A}_2(in_2)$ to denote that out_1 is the final output of algorithm $\mathcal{A}_1$ with local input in_1 after interacting with algorithm $\mathcal{A}_2$ with local input in_2.

2.1 Framework for Server-Aided Verification

Definition 1. *A signature scheme* (Setup, KeyGen, Sign, Verify) *with server-aided verification is one equipped with a* VStetup *algorithm and an* AidedVerify *protocol.*

- VStetup *is a probabilistic algorithm executed by a verifier which takes a security parameter* 1^λ *and outputs some verification parameter* VString.
- AidedVerify *is an interactive protocol between a verifier and a server. The input of the verifier includes* VString *generated by* VStetup, *a message/signature pair* (m, σ) *obtained from a signer and the public key* pk *of the signer. As a result of the protocol, the verifier outputs* $\top$ *iff* σ *is a valid signature on* m *under* pk, $\perp$ *otherwise. The server has no local output.*

In this paper we give a more fine-grained definition.

Definition 2. *Our definition of algorithms for server-aided verification are refined as below.*

1. VString *can be partitioned into private/public verification parameters* (τ, Γ)*, i.e.,* $(\tau, \Gamma) \xleftarrow{\$} \mathsf{VStetup}(1^\lambda)$.
2. AidedVerify *is split into a pair of interacting multi-stage algorithms* $\mathsf{AidedVerify}_c$ *and* $\mathsf{AidedVerify}_s$, *which denotes the respective part of the* AidedVerify *protocol executed by a client and a server.*

Definition 3. *Correctness of a signature scheme with server-aided verification requires that for all* $\lambda \in \mathbb{N}$, *all* Param *given by* $\mathsf{Setup}(1^\lambda)$, *all* $(\mathtt{sk}, \mathtt{pk})$ *given by* KeyGen(), *and all* VString *given by* $\mathsf{VStetup}(1^\lambda)$, *we have* $\mathsf{Verify}(\mathtt{pk}, \sigma, m) = \mathsf{AidedVerify}_c(m^*, \sigma^*, \mathtt{pk}, \mathtt{VString}) \leftrightarrow \mathsf{AidedVerify}_s(\Gamma)$ *holds for all* m *in the message space and* σ *in the signature space.*

2.2 Existential Unforgeability

The work of Wu *et al.* [13] gives two security definition. The first one is an extension of the existential unforgeability against adaptive chosen message attack of a standard signature scheme. Below gives our definition of unforgeability that will be used throughout this paper. This is based on the definition of Wu *et al.* with adjustments to be elaborated.

Definition 4. *A server-aided verification of signatures protocol is* (t, q_s, q_v)*-existentially unforgeable under adaptive chosen message and verification attacks (EUF-CMVA) if for all probabilistic polynomial time adversary* $\mathcal{A}$ *which runs within time* t *wins the following game with negligible probability in a security parameter* λ.

Setup. *The challenger* $\mathcal{C}$ *runs the algorithms* Setup, KeyGen *and* VStetup *to obtain the system parameter* Param, *a key pair* $(\mathtt{sk}, \mathtt{pk})$ *and the verification private/public parameter* $\mathtt{VString} = (\tau, \Gamma)$ *respectively. The adversary* $\mathcal{A}$ *is given* Param, pk *and* Γ, *while* τ *and* sk *are withheld from* $\mathcal{A}$.

Query. $\mathcal{A}$ *can adaptively make at most* q_s OSign *queries and at most* q_v OAidedVer *queries.*

- *In an* OSign *query,* $\mathcal{A}$ *supplies a message* m. $\mathcal{C}$ *puts* m *into a set* $\mathcal{M}$ *and responses with* $\mathsf{Sign}(\mathtt{sk}, m)$.
- *In an* OAidedVer *query,* $\mathcal{A}$ *starts by supplying a message/signature pair* (m, σ) *to* $\mathcal{C}$. $\mathcal{C}$ *then initiates the* AidedVerify *protocol as a client, and the adversary* $\mathcal{A}$ *responses as the server.*

Schemes	Sign	Verify
$\mathcal{BLS}$	$\sigma = H_0(m)^x$	$\hat{e}(g,\sigma) \stackrel{?}{=} \hat{e}(\mathtt{pk}, H_0(m))$
$\mathcal{ZSS}$	$\sigma = g^{\frac{1}{(H_1(m)+x)}}$	$\hat{e}(g,g) \stackrel{?}{=} \hat{e}(\sigma, g^{H_1(m)} \cdot \mathtt{pk})$
$\mathcal{BB}$	$(\sigma_1,\sigma_2) = (g^{\frac{1}{H_1(m)+x+yr}}, r)$	$\hat{e}(g,g) \stackrel{?}{=} \hat{e}(\sigma_1, g^{H_1(m)} \cdot \mathtt{pk}_X \cdot \mathtt{pk}_Y^{\sigma_2})$
$\mathcal{BGLS}$	$\sigma_i = H_0(m_i)^{x_i}$ $\sigma = \prod_{i=1}^k \sigma_i$	$\hat{e}(g,\sigma) \stackrel{?}{=} \prod_{i=1}^k \hat{e}(\mathtt{pk}_i, H_0(m_i))$

Table 1: Concise Review of Four Signature Schemes

Output. *$\mathcal{A}$ outputs a message/signature pair (m^*, σ^*) to $\mathcal{C}$. $\mathcal{A}$ is considered to win the game if $m^* \notin \mathcal{M}$ and $\top \leftarrow \mathsf{AidedVerify}_{\mathsf{c}}(m^*, \sigma^*, \mathtt{pk}, (\tau,\Gamma)) \leftrightarrow \mathsf{AidedVerify}_{\mathsf{s}}(\Gamma)$.*

2.3 Review of Pairing-based Signatures

Table 1 gives a concise review of a few selected short signature schemes and an aggregate signature scheme. H_0, H_1 are cryptographic (collision resistant) hash functions which map arbitrary bit-strings to $\mathbb{G}$ and $\mathbb{Z}_p$ respectively. The private signing key is a random element $x \in_R \mathbb{Z}_p$; any additional component which may be required is denoted by $y \in_R \mathbb{Z}_p$. $\mathcal{BB}$ is provably secure in the standard model.

3. OUR SECURITY DEFINITION

We propose a better model of the security of a SAV scheme against collusion between the server and the signer.

DEFINITION 5. *A server-aided verification protocol is (t, q_v)-sound against adaptive chosen verification attacks under collusion if for all probabilistic polynomial time adversary $\mathcal{A}$ which runs within time t wins the following game with negligible probability in a security parameter λ.*

Setup. *The challenger $\mathcal{C}$ runs the algorithms Setup, KeyGen and VStetup to obtain the system parameter $\mathtt{Param}$, a key pair $(\mathtt{sk}, \mathtt{pk})$ and the private/public verification parameter $\mathtt{VString} = (\tau, \Gamma)$ respectively. The adversary $\mathcal{A}$ is given $\mathtt{Param}$, $(\mathtt{sk}, \mathtt{pk})$ and Γ, but not τ.*

Query. *$\mathcal{A}$ can adaptively make at most q_v OAidedVer queries, as defined in Definition 4. $\mathcal{A}$ can maintain some state information $\mathtt{state}$ throughout all these queries.*

Output. *With $\mathtt{state}$, $\mathcal{A}$ outputs a message m^** and a signature σ^* *to $\mathcal{C}$. Let $b_0 \leftarrow \mathsf{Verify}(\mathtt{pk}, \sigma^*, m^*)$ and $b_1 \leftarrow \mathsf{AidedVerify}_{\mathsf{c}}(m^*, \sigma^*, \mathtt{pk}, \mathtt{VString}) \leftrightarrow \mathcal{A}(\mathtt{state})$. We say that $\mathcal{A}$ wins if $b_0 \neq b_1$.*

An alternative definition can be given by assigning $b_0 \leftarrow \mathsf{AidedVerify}_{\mathsf{c}}(m^*, \sigma^*, \mathtt{pk}, \mathtt{VString}) \leftrightarrow \mathsf{AidedVerify}_{\mathsf{s}}(\Gamma)$ instead. It is equivalent to the above definition by the correctness of a SAV protocol (Definition 3).

4. OUR SOLUTION

The basic idea of our protocol is similar to the generic protocol of Girault and Lefranc [9]. Let f be a scheme-specific public function which takes ($\mathtt{Param}$, $\mathtt{pk}$, m, σ) as input. Their protocol assumes the predicate is in the form of $\hat{e}(g,g) = \hat{e}(\sigma', f(\mathtt{Param}, \mathtt{pk}, m, r))$, where $\sigma = (\sigma', r)$, σ' is the group element in σ and r is the other part of the signature, if any[1]. Hence the application is rather limited, e.g., it cannot be applied on schemes which involve predicates not based on $\hat{e}(g,g)$ [3, 2]. In contrast, our protocol works with predicate which is based on the comparisons between elements in $\mathbb{G}_T$ or products of them. This generalizes the predicate class of [9] and effectively covers nearly all pairing-based verification algorithms, in particular, signature schemes such as [2, 3, 6] studied in [8].

4.1 Our Generic Protocol

We first establish some notations. Let $\mathcal{F}$ be a scheme-specific function which takes ($\mathtt{Param}$, $\mathtt{pk}$, m, σ) as input and let $\mathcal{F}_{agg}$ be a function that is specific to an aggregate signature scheme which takes $\mathtt{Param}$, $\{\mathtt{pk}_i\}$, $\{m_i\}$, σ as input. In the basic form, the output of $\mathcal{F}$ or $\mathcal{F}_{agg}$ only consists of a set of group-$\mathbb{G}$ elements $\{C_i\}_{i\in[1,\nu]}$ for a positive integer ν. A more general definition $\mathcal{F}'$ or $\mathcal{F}'_{agg}$ allows the output to be pairs of a group-$\mathbb{G}$ element and an exponent in $\mathbb{Z}_p$, $\{(C_i, v_i)\}_{i\in[1,\nu]}$. Let $\mathcal{V}$ be a predicate in the form of $\prod_{(i,j)\in\mathcal{X}} \hat{e}(C_i, C_j) \stackrel{?}{=} \prod_{(k,l)\in\mathcal{Y}} \hat{e}(C_k, C_l)$ for some subsets $\mathcal{X}, \mathcal{Y} \subseteq \{1,\cdots,\nu\} \times \{1,\cdots,\nu\}$. Similarly, corresponding to the more general definition $\mathcal{F}'$ or $\mathcal{F}'$, let $\mathcal{V}'$ be a predicate in the form of $\prod_{(i,j)\in\mathcal{X}} \hat{e}(C_i^{v_i}, C_j^{v_j}) \stackrel{?}{=} \prod_{(k,l)\in\mathcal{Y}} \hat{e}(C_k^{v_k}, C_l^{v_l})$ for some subsets $\mathcal{X}, \mathcal{Y} \subseteq \{1,\cdots,\nu\} \times \{1,\cdots,\nu\}$.

We reduce the task of server-aided verification to the task of "secure" delegation of pairing computation. Let $\mathfrak{S}$ be a secure pairing delegation protocol to be described in details. The framework of our generic protocol is given below.

1. The verifier runs the pre-computation phase of $\mathfrak{S}$, possibly with the knowledge of a (set of) public key.
2. After receiving a (set of) message, a signature and a (set of) public key, the verifier executes $\mathcal{F}$ (or $\mathcal{F}_{agg}$) to obtain $\{C_i\}_{i\in[1,\nu]}$ (or $\{(C_i, v_i)\}_{i\in[1,\nu]}$ for $\mathcal{F}$ or $\mathcal{F}'_{agg}$).
3. The verifier engages with the server in the request-response phase of $\mathfrak{S}$ to ask for the values of $\hat{e}(C_i, C_j)$, $\hat{e}(C_k, C_l)$ where $(i,j) \in \mathcal{X}, (k,l) \in \mathcal{Y}$ according to the predicate $\mathcal{V}$ (or $\mathcal{V}'$).
4. The verifier runs the verification phase of $\mathfrak{S}$ to ensure the pairing are computed correctly.
5. The verifier finally checks if the predicate is true according to the values obtained in the output phase of $\mathfrak{S}$, which is gives the result of the verification of the signature.

4.2 Pairing Delegation Protocols

The key element of our SAV protocol is pairing delegation $\mathfrak{S}$, which is introduced in [4]. Below we adopt slightly the definition from [11].

DEFINITION 6. *A pairing delegation protocol is a two-move interactive protocol between a client (delegator) and a server (delegatee) consists of five phases – pre-computation, request, response, verification and output.*

1. *Pre-computation phase: the client performs local computation taking the system parameter and possibly one of the points A, B as the input. The system parameter includes the bilinear map context $(\mathbb{G}, \mathbb{G}_T, g, p, \hat{e})$.*

[1]We abused the notation of f a bit such that it may take a truncated part of the input (only a part of σ in their case).

2. *Request phase: the client sends a message to the server based on the output of the pre-computation phase. In case that one of the points is not used in the pre-computation phase, The message to be sent should be further processed with the knowledge of any points which are not used in the pre-computation phase.*

3. *Response phase: the server sends a message to the client. The input of the server includes the system parameter and the message from the client in the request phase.*

4. *Verification phase: the client verifies the message from the server based on the output of the pre-computation phase. It halts and output $\perp$ if the check fails.*

5. *Output phase: the client outputs a $\mathbb{G}_T$ element $\hat{e}(A, B)$ and the server has no local output.*

To our knowledge, the studies of pairing delegation protocols in the literature include [4, 10, 11] . Only the protocols in [11] consider batch delegation.

DEFINITION 7 ([11]). *Batch pairing delegation protocol is an interactive protocol between a client (delegator) and a server (delegatee). The input of the client consists of a set of points $A, \{B_i\}_{i\in[1,\nu]} \in \mathbb{G}$ for a positive integer ν. As a result of the protocol, the client outputs a set of $\mathbb{G}_T$ elements $\{\hat{e}(A, B_i)\}_{i\in[1,\nu]}$ and the server has no local output.*

The first reason we turn our attention to the batch pairing delegation protocol is that, the definitions of soundness in [4, 10] do not specify whether A and/or B are known to the adversary (the malicious server in our context). Indeed, none of the protocols in [10] is designed for the case which both of the two input points are public. This is important in SAV since the input of the verification algorithm is by definition public (also recall that the adversary in our adversarial model is equipped with the message, the signature, the system parameters, etc.) It is simply not suitable for our purpose if the security of the protocols relies on the fact that the adversary does not know the input of the predicate. Moreover, it has been suggested [11] that there exists a delegation protocol that is secure if and only if A is private from a simplification of an existing protocol [10]. Finally, the protocol for public points in [11] is more efficient than that its counterpart in [4]. For these reasons, we choose to instantiate $\mathfrak{S}$ with the protocol PVPC (an acronym for "Public Variable and Public Constant") [11] which we review below.

- **Pre-computation.** The client first picks a generator $Q \in \mathbb{G}$ and a random exponent $r_Q \in_R \mathbb{Z}_p$; then computes $\tilde{Q} = Q^{r_Q}$ and $\hat{e}(A, \tilde{Q}) = \hat{e}(A, Q)^{r_Q}$.

- **Request.** The client sends to the server $\langle A, B_0 = \tilde{Q} \cdot \prod_{i=1}^{n} B_i^{b_i}, B_1, \ldots, B_n\rangle$, where $b_i \in_R \mathbb{Z}_p$.

- **Response.** The server sends $\langle \alpha_0, \alpha_1, \ldots, \alpha_n\rangle$, where $\alpha_i = \hat{e}(A, B_i)$.

- **Output.** The client verifies if $\alpha_i \in \mathbb{G}_T$ for $i = 0$ to n and $\hat{e}(A, \tilde{Q}) \prod_{i=1}^{n} \alpha_i^{b_i} = \alpha_0$ all hold. If so, outputs $\{\alpha_i\}$, $\perp$ otherwise.

Limitations. A more flexible type of pairing delegation protocol is PVPV (an acronym for "Public Variable and Public Variable") which is not available to the best of our knowledge. The other limitation is that the knowledge of the point A is required in the above PVPC protocol. We remark that there exists non-batch PVPC protocol which does not require the knowledge of the constant A in the precomputation stage [4].

4.3 Our SAV Protocol for Short Signatures

Following our generic protocol, we define function $\mathcal{F}_{\mathcal{BLS}}$ and predicate $\mathcal{V}_{\mathcal{BLS}}$ to build a SAV protocol for Boneh-Lynn-Shacham signature [3], with the notation in Table 1.

- Function $\mathcal{F}_{\mathcal{BLS}} : (\mathtt{Param}, \mathtt{pk}, m, \sigma) \mapsto (g, \sigma, \mathtt{pk}, H_0(m))$.

- Predicate $\mathcal{V}_{\mathcal{BLS}}$: $\hat{e}(C_1, C_2) \stackrel{?}{=} \hat{e}(C_3, C_4)$ with $C_1 = g$, $C_2 = \sigma$, $C_3 = \mathtt{pk}$, $C_4 = H_0(m)$.

The verifier engages in the pairing delegation protocol $\mathfrak{S}$ with the server and obtains the two pairing values $\alpha_0 = \hat{e}(g, \sigma)$ and $\alpha_1 = \hat{e}(\mathtt{pk}, H_0(m))$. Finally, the verifier checks if the predicate is true, that is, whether $\alpha_0 \stackrel{?}{=} \alpha_1$.

Since the pairing delegation protocol $\mathfrak{S}$ we employ is capable of delegating pairing computations in batch, the verifier can offload the verification of many signatures from the same signer to further reduce the computation overhead. Specifically, the verifier can delegate the computation of $\{\alpha_{0i} = \hat{e}(g, \sigma_i)\}$ and $\{\alpha_{1i} = \hat{e}(\mathtt{pk}, H_0(m_i))\}$ in batch.

Unfortunately, using a PVPC protocol instead of a PVPV protocol, the pairing of $\hat{e}(C_1, C_2)$ and $\hat{e}(C_3, C_4)$ cannot be batched since only $C_1 = g$ is a constant.

We can define functions $\mathcal{F}'_{\mathcal{ZSS}}$, $\mathcal{F}'_{\mathcal{BB}}$ and predicates $\mathcal{V}'_{\mathcal{ZSS}}$, $\mathcal{V}'_{\mathcal{BB}}$ for $\mathcal{ZSS}$ [14], and Boneh-Boyen [1] as below. Note that we omit the exponent v_i when it is equal to 1 for brevity.

- Function $\mathcal{F}'_{\mathcal{ZSS}}$:
  $(\mathtt{Param}, \mathtt{pk}, m, \sigma) \mapsto ((g, H_1(m)), \sigma, \mathtt{pk}, \sigma, g, g)$.

- Predicate $\mathcal{V}'_{\mathcal{ZSS}}$: $\hat{e}(C_1^{v_1}, C_2)\hat{e}(C_3, C_4) \stackrel{?}{=} \hat{e}(C_5, C_6)$ with $C_1 = g$, $v_1 = H_1(m)$, $C_2 = \sigma$, $C_3 = \mathtt{pk}$, $C_4 = \sigma$, $C_5 = g$, $C_6 = g$.

- Function $\mathcal{F}'_{\mathcal{BB}} : (\mathtt{Param}, (\mathtt{pk}_X, \mathtt{pk}_Y), m, (\sigma_1, \sigma_2)) \mapsto$
  $((g, H_1(m)), \sigma_1, \mathtt{pk}_X, \sigma_1, (\mathtt{pk}_Y, \sigma_2), \sigma_1, g, g)$.

- Predicate $\mathcal{V}'_{\mathcal{BB}}$:
  $\hat{e}(C_1^{v_1}, C_2)\hat{e}(C_3, C_4)\hat{e}(C_5^{v_5}, C_6) \stackrel{?}{=} \hat{e}(C_7, C_8)$ with $C_1 = g$, $v_1 = H_1(m)$, $C_2 = \sigma_1$, $C_3 = \mathtt{pk}_X$, $C_4 = \sigma_1$, $C_5 = \mathtt{pk}_Y$, $v_5 = \sigma_2$, $C_6 = \sigma_1$, $C_7 = g$, $C_8 = g$.

In both $\mathcal{ZSS}$ and $\mathcal{BB}$, we have the evaluation of $\hat{e}(g, g)$ which is a constant independent of the signer's public key or the signature. The client can simply store it without re-computing it every time.

Comparison with Batch Signature Verification. The concepts of using batch pairing delegation for verifying many signatures and batch signature verification look almost the same at the first glance. However, there are subtle differences. While the computation of pairing are delegated in batch, the pairing values are not aggregated together and the client can get each of the individual pairing value. In other words, the client is able to verify each of the signature by simple equality checks (and possibly with exponentiations in $\mathbb{G}_T$). On the other hand, extra measures are usually needed to identify the invalid signature(s) in batch verification.

With respect to concrete instantiations available today, for the case of batching the signatures form different n signers, $(n+1)$ pairing computations are needed by the batch verifier for $\mathcal{BLS}$ proposed in [8]. In our approach, these pairings are delegated, and the client only requires to do $O(n)$ of exponentiations. In the batch verifier approach [8], $O(n)$ of exponentiations only involve small exponents, which may also be used in the batch pairing delegation protocol [11] since secrecy of the points are not a concern here.[2]

Using the PVPC protocol [11] in verifying n signatures from the same signer produced by $\mathcal{ZSS}$ or $\mathcal{BB}$ require 2 or 3 times of pairing computations (at the server side) respectively than the normal approach. The cost can be justified when the batch PVPC protocol is faster than n invocations of the non-batch protocol by a factor of 3. Lastly, we remark that we are not aware of any batch verifier for $\mathcal{ZSS}$ or $\mathcal{BB}$.

Finally, we remark that these two approaches can be used together, i.e., first identify what are the required pairing computations in a batch verifier, then these computations can still be delegated by using a pairing delegation protocol.

4.4 Our Protocol for Aggregate Signatures

For an aggregated signature σ, a list of messages $\{m_i\}_{i=1}^n$ and the public key list of the corresponding signers $\{\mathtt{pk}_i\}_{i=1}^n$, we define function $\mathcal{F}_{agg}$ and predicate $\mathcal{V}$ to build a SAV protocol for Boneh *et al.*'s aggregate signature [2].

- Function $\mathcal{F}_{agg}$: $(\mathtt{Param}, \mathtt{pk}_1, \ldots, \mathtt{pk}_n, m_1, \ldots, m_n, \sigma) \mapsto (g, \sigma, \mathtt{pk}_1, H_0(m_1), \ldots, \mathtt{pk}_n, H_0(m_n))$.
- Predicate $\mathcal{V}_{agg}$: $\hat{e}(C_1, C_2) \stackrel{?}{=} \prod_{i=1}^n \hat{e}(C_{2i+1}, C_{2i+2})$ with $C_1 = g$, $C_2 = \sigma$, $C_{2i+1} = \mathtt{pk}_i$, $C_{2i+2} = H_0(m_i)$ for $i = 1$ to n.

The verifier engages in the pairing delegation protocol $\mathfrak{S}$ with the server and obtains the pairing values $\alpha_0 = \hat{e}(g, \sigma)$ and $\alpha_i = \hat{e}(\mathtt{pk}_i, H_0(m_i))$ for $i = 1$ to n. Finally, the verifier checks if the predicate is true, that is, whether $\alpha_0 \stackrel{?}{=} \prod_{i=1}^n \alpha_i$. Again, we remark that we are not aware of any protocol of type PVPV and hence the delegation of these pairing computations cannot be batched together. On the other hand, computation overhead can be reduced further by batch delegation of pairing values for the same signer.

4.5 Security Analysis

Security Properties of Pairing Delegation. Two security requirements of pairing delegation protocol include completeness and correctness. Informally, completeness means that the client can obtain the correct values after interacting with an honest server; and correctness means that the client can detect (with high probability) when the server is cheating, which means the final result consists of a wrong value. The formal definitions can be found in [11].

Security of Our Protocol. Existential unforgeability follows from the completeness of $\mathfrak{S}$ and the unforgeability of the underlying signature scheme. If a forgery passes the verification algorithm of the underlying signature, i.e., it satisfies the predicate determined by the verification algorithm, by the completeness of $\mathfrak{S}$, the delegated predicate will evaluate to the same value, and hence the forgery will pass (an honest execution of) AidedVerify.

Soundness also follows readily from the soundness of $\mathfrak{S}$. If the delegated evaluation of the predicate mismatches with a direct evaluation of the predicate, there must exist one wrongly-computed pairing value that is undetected by the client, which breaks the soundness of $\mathfrak{S}$. This can always be identified by simply recomputing all the pairings involved which can be done in a polynomial time.

[2]One can use small exponents $\{b_i\}$ in the PVPC protocol. However, at least one of them should be chosen from a large domain. Otherwise an adversary may guess the hidden $\tilde{Q}$ correctly and break the protocol.

5. REFERENCES

[1] Dan Boneh and Xavier Boyen. Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. *J. Cryptology*, 21(2):149–177, 2008.

[2] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *EUROCRYPT*, pages 416–432, 2003.

[3] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. *J. Cryptology*, 17(4):297–319, 2004.

[4] Benoit Chevallier-Mames, Jean-Sebastien Coron, Noel McCullagh, David Naccache, and Michael Scott. Secure Delegation of Elliptic-Curve Pairing. Cryptology ePrint Archive, 2005/150, 2005.

[5] Sherman S. M. Chow, Joseph K. Liu, and Jianying Zhou. Identity-based online/offline key encapsulation and encryption. In *ASIACCS*, 2011. To appear.

[6] Sherman S. M. Chow, Siu-Ming Yiu, and Lucas Chi Kwong Hui. Efficient Identity Based Ring Signature. In *ACNS*, pages 499–512, 2005.

[7] Cheng-Kang Chu, Wen Tao Zhu, Sherman S. M. Chow, Jianying Zhou, and Robert H. Deng. Secure mobile subscription of sensor-encrypted data. In *ASIACCS*, 2011. To appear.

[8] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical Short Signature Batch Verification. In *CT-RSA*, pages 309–324, 2009.

[9] Marc Girault and David Lefranc. Server-Aided Verification: Theory and Practice. In *ASIACRYPT*, pages 605–623, 2005.

[10] Bo Gyeong Kang, Moon Sung Lee, and Je Hong Park. Efficient Delegation of Pairing Computation. Cryptology ePrint Archive, Report 2005/259, 2005.

[11] Patrick P. Tsang, Sherman S. M. Chow, and Sean W. Smith. Batch Pairing Delegation. In *IWSEC*, pages 74–90, 2007.

[12] Zhiwei Wang, Licheng Wang, Yixian Yang, and Zhengming Hu. Comment on Wu et al.'s Server-Aided Verification Signature Schemes. *Intl. J. of Network Sec.*, 10(2):158–160, 2010.

[13] Wei Wu, Yi Mu, Willy Susilo, and Xinyi Huang. Server-Aided Verification Signatures: Definitions and New Constructions. In *ProvSec*, pages 141–155, 2008.

[14] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. An Efficient Signature Scheme from Bilinear Pairings and Its Applications. In *PKC*, pages 277–290, 2004.