Hindawi Publishing Corporation
Mobile Information Systems
Volume 2015, Article ID 626415, 11 pages
http://dx.doi.org/10.1155/2015/626415

Research Article

Server-Aided Verification Signature with Privacy for Mobile Computing

Lingling Xu,[1] Jin Li,[2] Shaohua Tang,[1] and Joonsang Baek[3]

[1]*School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China*
[2]*School of Computer Science and Educational Software, Guangzhou University, Guangzhou 510006, China*
[3]*Khalifa University of Science, Technology and Research, P.O. Box 127788, Abu Dhabi, UAE*

Correspondence should be addressed to Shaohua Tang; csshtang@scut.edu.cn

Received 6 May 2014; Accepted 1 September 2014

Academic Editor: David Taniar

With the development of wireless technology, much data communication and processing has been conducted in mobile devices with wireless connection. As we know that the mobile devices will always be resource-poor relative to static ones though they will improve in absolute ability, therefore, they cannot process some expensive computational tasks due to the constrained computational resources. According to this problem, server-aided computing has been studied in which the power-constrained mobile devices can outsource some expensive computation to a server with powerful resources in order to reduce their computational load. However, in existing server-aided verification signature schemes, the server can learn some information about the message-signature pair to be verified, which is undesirable especially when the message includes some secret information. In this paper, we mainly study the server-aided verification signatures with privacy in which the message-signature pair to be verified can be protected from the server. Two definitions of privacy for server-aided verification signatures are presented under collusion attacks between the server and the signer. Then based on existing signatures, two concrete server-aided verification signature schemes with privacy are proposed which are both proved secure.

1. Introduction

Recent advances in wireless technology have led to mobile computing [1, 2] which is a technology that enables access to digital resources at any time, from any location. In mobile computing, much data communication and processing is conducted in mobile devices with wireless connection such as cell-phones, security access-cards, and sensors. Therefore, mobile computing represents the elimination of time-and-place restrictions imposed by desktop computers and wired networks. As we know mobile devices must be light and small to be easily carried around. Such considerations, in conjunction with a given cost and level of technology, will exact a penalty in computational resources of mobile devices such as processor speed. While mobile devices will improve in absolute ability, they will always be computationally weak in relation to static ones. As a consequence there are tasks, which potentially could enlarge a device's range of application, which are beyond its reach. A natural solution is to outsource computations that are too expensive for one device, to other devices which are more powerful or numerous and connected to the device. For example, consider a sensor that is presented with an access-card, sends it a random challenge, and receives a digital signature of the random challenge. The computation is required to verify the signature involves public-key operations which are too expensive in both time and space for the sensor to run. Instead, it could outsource the verification to a powerful device in order to reduce its computational load. Recently, with the development of cloud computing, server-aided computation has received widespread attention which enables power-constrained devices to outsource expensive computational tasks to a server. The related works such as server-aided delegated computation [3–8] and server-aided verification signatures [9–16] have been widely studied. Delegated computation is a protocol between two polynomial-time parties, a client, and a server, to collaborate on the computation of a function F. Concretely, the client wants the server to compute $F(x)$ for any input instance x by

the delegated computation protocol and verify the correctness of the results that is returned by the server. A key requirement is that the amount of work performed by the client to generate and verify work instances must be substantially cheaper than performing the computation on its own.

A server-aided verification signature scheme consists of a digital signature scheme and a server-aided verification protocol. Signatures can be verified by executing the server-aided verification protocol with the server, where the verification requires less computation than the original verification algorithm of the digital signature. Different to delegated computation, the existing server-aided verification signature schemes can achieve the soundness of the server-aided verification protocol under their security definitions, namely, a trusted server cannot convince the verifier that an invalid signature is valid, and the verifier cannot directly verify the results computed by the server. The notion of server-aided verification signature was first introduced by Quisquater and de Soete [10] for speeding up RSA verification with a small exponent. Then, Lim and Lee [11] extended this idea into discrete-logarithm based schemes, by proposing efficient protocols for speeding up the verification of discrete-logarithm based identity proofs and signatures. Girault and Quisquater [13] introduced a different approach for server-aided verification signature which does not require precomputation or randomization. Its security remains computational, based on the hardness of a subproblem (viz. factorization) of the initial underlying problem (viz. composite discrete logarithm). Hohenberger and Lysyanskaya [17] addressed the situation in which the server is made of two untrusted softwares, which are assumed not to communicate with each other. Girault and Lefranc [14] presented a generic server-aided verification protocol for digital signatures from bilinear maps which has been used to construct many digital signature schemes such as [18–23].

As to the security of server-aided verification signature, many efforts have been devoted to defining strong security models for it. The schemes [10, 11, 13, 14] considered the security property based on the assumption that the malicious server does not have any valid signatures on the message when it tries to prove an invalid signature of that message to be valid. Among them, the scheme [13] is computationally secure based on the hardness of a subproblem of the underlying complexity problem in the original signature scheme. To give stronger definition of this property, Wu et al. [15] formally defined this security assuming that the malicious server may collude with the signer and obtain the secret key of the signer. They first introduced and defined the existential unforgeability of server-aided verification signatures and considered collusion between a signer and a server, who collaboratively prove an invalid signature to be valid. In addition, under their security models, they introduced the server-aided verification for the Waters signature [21] and the BLS signature [18], respectively.

Though the existing server-aided verification signature schemes above have been devoted many efforts to their security models, they only considered the soundness to protect the malicious server who may try to prove an invalid signature of a message to be valid. However, in some applications where the message-signature to be verified contains some sensitive information, for example, the message contains important business secrets or is related to medical information, the verifier does not want the server learn anything about the message and/or the signature to protect its privacy. So, the message privacy of the server-aided verification protocol is also desired besides the soundness. Though in Wu et al. [15], based on Waters Signature [21] and BLS signature [18], two SA verification signature schemes (see Section 4 in [15]) were presented in which the message to be verified is not revealed to the server, the schemes cannot achieve the soundness under collusion and adaptively chosen message attacks.

In this paper, we will present two privacy definitions for server-aided verification signature under collusion by the server and the signer and adaptive chosen message attacks. A server-aided verification signature scheme with privacy also consists of a digital signature scheme and a server-aided verification protocol.

(1) The first privacy definition for the server-aided verification signature is about message privacy; namely, the server cannot learn anything about the message to be verified during the server-aided verification protocol even if it possesses the secret key of the signer. Generally, when the verifier wants the server to verify a message-signature pair, it will "blind" this message at the beginning of the server-aided verification protocol so that the server cannot obtain any information about this message, while it can verify the validity of the message-signature pair by using the server's responses.

(2) The second privacy definition for the server-aided verification signature is about message-signature privacy which is stronger than the first one, and in this definition, the server can learn nothing about the message-signature pair to be verified even if it colludes with the signer. To achieve this privacy, similarly, the verifier will "blind" the message-signature pair at the beginning of the server-aided verification protocol so that the server cannot obtain any information about the message or the signature; however it can verify the validity of the message-signature pair after the server responds.

For the two privacy notions, we present detailed and strict security models. Then, under the security models, we present two concrete constructions for server-aided verification signature based on Waters signature [21] and BLS signature [18] which, respectively, achieve message privacy and message-signature privacy. The soundness of the two constructions is proved under the strong definition of [15] assuming that the malicious server may collude with the signer and obtain the secret key of the signer. In addition, the efficiency analysis of the server-aided verification protocols shows that our two concrete server-aided verification signature schemes are both computation saving. Computation saving is probably the most obvious property that can distinguish a server-aided verification signature scheme SAV-Σ from an ordinary signature scheme Σ. This property enables the verifier in SAV-Σ

to check the validity of signatures in a more computationally efficient way than that in Σ.

Organization. This paper is organized as follows. In Section 2, we will review some fundamental backgrounds, the definition of server-aided verification signatures and the security notions defined in [15] including existential unforgeability and soundness against collusion and adaptive chosen message attacks. In Section 3, we will present the message privacy of server-aided verification signatures, give a concrete construction based on Waters signature scheme, and prove its security under our security model for message privacy. In Section 4, a stronger privacy of server-aided verification signatures named message-signature privacy will be defined and a provably secure concrete construction will be presented based on BLS signature scheme. Finally we conclude in Section 5.

2. Preliminaries

2.1. Syntax. Throughout the paper, if A is a randomized algorithm, then $y \leftarrow \mathsf{A}(x)$ denotes the assignment to y of the output of A on input x. Unless noted, all algorithms are probabilistic polynomial-time (PPT) and we implicitly assume that they take an extra parameter κ in their input, where κ is a security parameter.

2.2. Bilinear Maps. Let $\mathbb{G}_1, \mathbb{G}_T$ be two (multiplicative) cyclic groups such that $|\mathbb{G}_1| = |\mathbb{G}_T| = p$, where p is a large prime. Let g be a generator of $\mathbb{G}_1$, and e be an admissible bilinear map: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$, satisfying (1) for all $a, b \in \mathbb{Z}_p$; it holds that $e(g^a, g^b) = e(g, g)^{ab}$; (2) $e(g, g) \neq 1$; and (3) it is efficiently computable.

We say that $(\mathbb{G}_1, \mathbb{G}_T)$ are bilinear groups if there exists the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ as above, and the group action in $\mathbb{G}_1$ and $\mathbb{G}_T$ can be computed efficiently. Such groups can be built from Weil pairing or Tate pairing on elliptic curves.

2.3. Server-Aided Verification Signature. A server-aided verification signature scheme SAV-Σ consists of six algorithms: ParamGen, KeyGen, Sign, Verify, SA-Verifier-Setup, and SA-Verify. The first four algorithms are the same as those in an ordinary signature scheme Σ. SAV-Σ contains three parties, respectively, a signer, a verifier, and a server.

(i) $param \leftarrow$ ParamGen. This algorithm takes a security parameter κ and returns a string $param$ as input, which denotes the common scheme parameters, including the description of the message space $\mathbb{M}$ and the signature space Ω.

(ii) $(pk, sk) \leftarrow$ KeyGen($param$). This algorithm takes $param$ as input and outputs a key pair (pk, sk), where sk is the signing key and pk is the verification key.

(iii) $\sigma \leftarrow$ Sign($param, m, sk, pk$). The signer takes a message $m \in \mathbb{M}$, the system parameter $param$ and the key pair (sk, pk) as inputs, outputs a signature σ.

(iv) {Valid, Invalid} $\leftarrow$ Verify($param, m, \sigma, pk$). The verifier takes the parameter $param$, a message-signature pair (m, σ) and the public key pk, outputs Valid/Invalid to indicate that σ is a valid/invalid signature on m under pk.

(v) VString $\leftarrow$ SA-Verifier-Setup($param$). The verifier takes as input the system parameter $param$ and outputs a string VString which contains the information which can be precomputed by it.

(vi) {Valid, Invalid} $\leftarrow$ SA-Verify(Server$^{(param)}$, Verifier$^{(m,\sigma,pk,\mathsf{VString})}$). This is an interactive protocol between the server and the verifier where the server takes $param$ as input and the verifier takes $(m, \sigma, pk, \mathsf{VString})$ as inputs. Finally, the verifier outputs Valid if the server can convince it that σ is a valid signature on m. Otherwise, the verifier outputs Invalid.

In a SA verification signature scheme, we assume that the verifier has a limited computational ability and is not able to perform all computations in Verify alone. So, a SA verification signature scheme must satisfy an important property called computation saving property, which requires that the computations performed by the verifier in SA-Verify must be less than those performed in Verify.

2.4. Security Model for Server-Aided Verification Signature. In the following, we will first present the security model for SAV-Σ with message privacy. As for the existential unforgeability of SAV-Σ, we will adopt existential unforgeability of SAV-Σ defined in [15], including the existential unforgeability against adaptive chosen message attacks of Σ defined in [24] and the soundness against collusion and adaptive chosen message attacks of SA-Verify. In the following, we will present the existential unforgeability of SAV-Σ as [15]. It requires that the adversary should not be (computationally) capable of producing a signature of a new message which can be proved as valid by SA-Verify, even if the adversary acts as a server.

Definition 1 (existential unforgeability against adaptive chosen message attacks of Σ). The adversary $\mathscr{A}$ and the challenger $\mathscr{C}$ play the following game.

(i) *Setup*. The challenger $\mathscr{C}$ runs the algorithms ParamGen and KeyGen to obtain system parameter $param$ and one key pair (sk, pk). The adversary $\mathscr{A}$ is given $param$ and pk.

(ii) *Queries*. The adversary $\mathscr{A}$ is allowed to make at most q_s sign queries. For each sign query $m_i \in \{m_1, \ldots, m_{q_s}\}$, the challenger $\mathscr{C}$ returns $\sigma_i =$ Sign($param, m_i, sk, pk$) as the response.

(iii) *Output*. Eventually, the adversary $\mathscr{A}$ outputs a pair (m^*, σ^*) and wins the game if:

(1) $m^* \notin \{m_1, \ldots, m_{q_s}\}$;

(2) Verify($param, m^*, \sigma^*, pk$) = Valid.

An adversary $\mathcal{A}$ is said to (t, q_s, ε)-break a signature scheme Σ if $\mathcal{A}$ runs in time at most t and makes at most q_s signature queries and the success probability $\Sigma - \mathsf{Adv}_{\mathcal{A}}$ to win the game above is at most ε.

We say that Σ is existentially unforgeable against adaptive chosen message attacks if there exists an adversary that (t, q_s, ε)-breaks it.

In the following, we will present the soundness against collusion and adaptive chosen message attacks of SA-Verify which means that the server cannot prove an invalid signature to be valid even if it colludes with the signer.

Definition 2 (soundness against collusion and adaptive chosen message attacks of SA-Verify). The adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ play the following game.

(i) *Setup*. The challenger $\mathcal{C}$ runs the algorithms ParamGen, KeyGen and SA-Verifier-Setup to obtain the system parameter *param*, one key pair (*sk*, *pk*) and VString. The adversary $\mathcal{A}$ is given *param* and (*sk*, *pk*).

(ii) *Queries*. Proceeding adaptively, the adversary $\mathcal{A}$ is allowed to make at most q_v server-aided verification queries. The challenger $\mathcal{C}$ responds by executing SA-Verify with the adversary $\mathcal{A}$, where the adversary $\mathcal{A}$ acts as the server and the challenger $\mathcal{C}$ acts as the verifier. At the end of each execution, the challenger returns the output of SA-Verify to the adversary $\mathcal{A}$.

(iii) *Output*. Eventually, the adversary $\mathcal{A}$ outputs a message m^*. The challenger $\mathcal{C}$ chooses a random invalid signature σ^* on the message m^*. Namely, it chooses a random element σ^* in $\Omega \setminus \Omega_{m^*}$, where Ω and Ω_{m^*} are, respectively, the signature space and the set of valid signatures of m^*. We say that $\mathcal{A}$ wins the game if

$$\mathsf{SA\text{-}Verify}\left(\mathcal{A}, \mathcal{C}^{(m^*, \sigma^*, pk, \mathsf{VString})}\right) = \mathsf{Valid}. \quad (1)$$

An adversary $\mathcal{A}$ is said to (t, q_v, ε)-break SA-Verify's soundness against collusion and chosen message attacks if $\mathcal{A}$ runs in time at most t, makes at most q_v server-aided verification queries and the success probability $\mathrm{Adv}_{\mathcal{A}}^{EU}$ to win the game above is at least ε.

We say that SA-Verify is (t, q_v, ε)-sound against collusion and chosen message attacks if there exists no adversary that (t, q_v, ε)-breaks it.

3. Server-Aided Verification Signature with Message Privacy

In this section, we will present the definition of message privacy for SA-Verify, and then, based on Waters signature scheme [21], present a concrete server-aided verification scheme with this privacy property. This privacy property is called *message privacy against collusion and adaptive chosen message attacks*. In this definition, the server is allowed to collude with the signer. Concretely, the server can obtain the key pair (*pk*, *sk*) of the signer and therefore can create the signature on any message. In addition, we will assume that the server cannot obtain the message-signature pairs that have been created by the signer before, alternatively, the signer will not store any message-signature pair that it has created. (Actually, this can be achieved by performing blind signature scheme presented in [25] between the signer and the verifier instead of performing the ordinary signature scheme. After the blind signature scheme, the verifier can obtain the ordinary message-signature pair without the signer learning anything about this pair. Then the verifier lets the server to verify the message-signature pair by performing SA-Verify. In this sense, even if the server colludes with the signer, it cannot obtain more information about the signed messages from the signer than it can obtain on its own. To clarify our privacy definition below more clearly, we simply assume that the server cannot obtain any message-signature pair which the signer has created for the verifier before.)

3.1. Definition of Message Privacy. A server-aided verification signature scheme with message privacy SAV-Σ also consists of six algorithms: ParamGen, KeyGen, Sign, Verify, SA-Verifier-Setup, and SA-Verify. The following is the definition of message privacy for the server-aided verification protocol under the collusion and adaptive chosen message attacks. In this definition, the server cannot obtain any information about the message to be verified under the collusion and adaptive chosen message attacks.

Definition 3 (message privacy of SA-Verify). We say that SA-Verify satisfies (t, q_v, ε)-message privacy against collusion and adaptive chosen message attacks if there exists no adversary $\mathcal{A}$ who runs in time at most t, makes at most q_v server-aided verification queries, and succeeds with probability at least ε in the following game with the challenger $\mathcal{C}$. The game is defined as follows.

(i) *Setup*. The challenger $\mathcal{C}$ runs the algorithms ParamGen, KeyGen and SA-Verifier-Setup to obtain system parameter *param*, one key pair (*sk*, *pk*), and VString. The adversary $\mathcal{A}$ is given *param* and (*sk*, *pk*). Note that $\mathcal{A}$ can generate any message-signature pair with the secret-public key pair (*sk*, *pk*); however as we assumed, it cannot obtain any message-signature pair that has been created by the signer before.

(ii) *Queries*. Proceeding adaptively, the adversary $\mathcal{A}$ is allowed to make at most q_v server-aided verification queries. The challenger $\mathcal{C}$ responds by executing SA-Verify with the adversary $\mathcal{A}$, where the adversary $\mathcal{A}$ acts as the server and the challenger $\mathcal{C}$ acts as the verifier. At the end of each execution, the challenger returns the output of SA-Verify to the adversary $\mathcal{A}$.

(iii) *Challenge*. $\mathcal{A}$ outputs two messages m_0, m_1, and sends them to the challenger $\mathcal{C}$. $\mathcal{C}$ chooses a bit $b \in \{0, 1\}$ at random and also chooses an element σ either randomly from Ω_{m_0} or randomly from Ω_{m_1}, where Ω_{m_0} and Ω_{m_1} are, respectively, the signature

space of m_0 and m_1. Then $\mathscr{C}$ and $\mathscr{A}$ interact with each other by running SA-Verify($\mathscr{A}, \mathscr{C}^{(m_b,\sigma,pk,\mathsf{VString})}$), where $\mathscr{A}$ plays as a server and $\mathscr{C}$ plays as a verifier. After the interaction, $\mathscr{C}$ sends the output of SA-Verify($\mathscr{A}, \mathscr{C}^{(m_b,\sigma,pk,\mathsf{VString})}$) to $\mathscr{A}$.

(iv) *Output*. Finally, $\mathscr{A}$ outputs a bit $b' \in \{0, 1\}$. We say that $\mathscr{A}$ wins the game with probability ε if

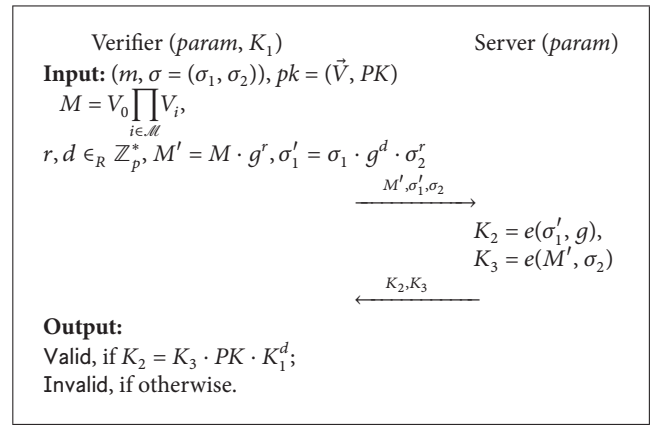
$$\Pr\left[b = b'\right] \geq \varepsilon. \tag{2}$$

Similar to Wu et al. [15], in the protocol Setup of the game above, VString is not provided to the adversary who now is acting as a server since VString might contain some private information of the verifier, which must be kept secret in server-aided verification signatures. In the definition, adversary $\mathscr{A}$ acts as the server and the challenger $\mathscr{C}$ acts as the verifier which will help $\mathscr{A}$ to extract some information from VString.

3.2. Concrete SA Verification Signature with Message Privacy. In the following, we will first present a concrete SA verification signature scheme with message privacy based on Waters signature [21]. The SA verification signature scheme with message privacy SAV-Σ consists of six algorithms: ParamGen, KeyGen, Sign, Verify, SA-Verifier-Setup, and SA-Verify. The first four algorithms are the same as those in Waters signature scheme [21]. As we know that, due to the elegant properties of pairing computation on elliptic curves, pairing has been widely employed as a building block for lots of cryptographic schemes, in particular in the construction of digital signatures. However, performing a pairing on an elliptic curve requires much more computational cost than executing both an exponentiation and a multiplication [16, 26–30], and for a power-constrained verifier who must execute multiple pairing computations during the verification of a message-signature pair, reducing the computational load of it is a meaningful task. In Waters signature [21], the verifier has to compute two pairings; however in SAV-Σ, its computational load is reduced and it will not compute any pairing. The concrete SA verification signature with message privacy based on Waters signature is described in detail as follows.

(i) $param \leftarrow$ ParamGen. Let κ be a security parameter, $(\mathbb{G}_1, \mathbb{G}_T)$ be bilinear groups where $|\mathbb{G}_1| = |\mathbb{G}_T| = p$ for some prime number $p \geq 2^\kappa$ and g be a generator of $\mathbb{G}_1$. $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ is a bilinear mapping. The system parameters are $param = (\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e)$ and the message space is $\mathbb{M} = \{0, 1\}^n$.

(ii) $(pk, sk) \leftarrow$ KeyGen($param$). Given the system parameters $(\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e)$, the signer chooses a random element $x \in \mathbb{Z}_p$, generates the public key pk as $(\vec{V}, PK)$ and $sk = x$ where $\vec{V}$ is a vector consisting of $n + 1$ elements $V_0, V_1, \ldots, V_n$ randomly selected in $\mathbb{G}_1$ and $PK = e(g, g)^x$.



Algorithm 1: SA-Verify with message privacy based on Waters signature.

(iii) $\sigma \leftarrow$ Sign($param, m, sk, pk$). For an n-bit message $m \in \{0, 1\}^n$, let $\mathscr{M} \subseteq \{1, 2, \ldots, n\}$ be the set of all i for which the ith bit of m is 1. The signer selects a random element $t \in \mathbb{Z}_p$ and generates the signature σ as $(\sigma_1, \sigma_2) = (g^x (V_0 \prod_{i \in \mathscr{M}} V_i)^t, g^t)$.

(iv) {Valid, Invalid} $\leftarrow$ Verify($param, m, \sigma, pk$). The verifier takes as input a claimed message-signature pair (m, σ), and outputs Valid if and only if $e(\sigma_1, g) = PK \cdot e(V_0 \prod_{i \in \mathscr{M}} V_i, \sigma_2)$. Otherwise it outputs Invalid.

(v) VString $\leftarrow$ SA-Verify-Setup($param$). The verifier takes as inputs the system parameters $(\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e)$ and computes $K_1 = e(g, g)$ as VString.

(vi) {Valid, Invalid} $\leftarrow$ SA-Verify(Server$^{(param)}$, Verifier$^{(m,\sigma,pk,\mathsf{VString})}$). This is an interactive protocol between the server and the verifier which is shown in Algorithm 1.

(1) Verifier, for a message-signature pair $(m, \sigma = (\sigma_1, \sigma_2))$ to be verified, first computes $M = V_0 \prod_{i \in \mathscr{M}} V_i$; then selects randomly $r, d \in \mathbb{Z}_p^*$, and blinds the message by computing $M' = M \cdot g^r$, $\sigma_1' = \sigma_1 \cdot g^d \cdot \sigma_2^r$; finally sends $(M', \sigma_1', \sigma_2)$ to the verifier.

(2) Server computes $K_2 = e(\sigma_1', g)$, $K_3 = e(M', \sigma_2)$ and returns K_2, K_3 to the verifier.

(3) Verifier checks the equation $K_2 = K_3 \cdot PK \cdot K_1^d$, and outputs Valid if it holds, and otherwise outputs Invalid.

Correctness of SA-Verify. For a claimed message-signature pair (m, σ), when the verifier and the server are both honest, namely, the verifier correctly computes M' and σ_1' and the server correctly computes K_2 and K_3, then the verification equation $K_2 = PK \cdot K_3 \cdot K_1^d$ holds if the message-signature pair

(m, σ) is valid and otherwise does not hold. In the following, we denote $(\sigma_1, \sigma_2) = (g^x(V_0\prod_{i\in\mathscr{M}} V_i)^t, g^t)$:

$$\begin{aligned} K_2 &= e\left(\sigma_1', g\right) \\ &= e\left(\sigma_1, g\right) \cdot e\left(g, g\right)^d \cdot e\left(\sigma_2^r, g\right) \\ &= e\left(M, g\right)^t \cdot PK \cdot K_1^d \cdot e\left(g^{tr}, g\right) \\ &= e\left(M \cdot g^r, g\right)^t \cdot PK \cdot K_1^d \\ &= e\left(M', \sigma_2\right) \cdot PK \cdot K_1^d \\ &= K_3 \cdot PK \cdot K_1^d. \end{aligned} \tag{3}$$

In the following, by Theorems 4 and 5, we will show that our SA verification signature scheme above is secure under our security model; namely, the SA verification protocol described in Algorithm 1 is sound against collusion and adaptive chosen message attacks and also satisfies message privacy against collusion and adaptive chosen message attacks.

Since Waters signature scheme has been proved existentially unforgeable against adaptive chosen message attacks, in order to prove that our SA verification signature scheme above is secure, we need only to prove that the SA verification protocol described in Algorithm 1 satisfies soundness against collusion and adaptive chosen message attacks defined in Definition 2 and message privacy against collusion and adaptive chosen message attacks.

Theorem 4. *The SA verification protocol described in Algorithm 1 satisfies $(t, q_v, 1/(p-1))$-soundness against collusion and adaptive chosen message attacks.*

Proof. In order to prove that the SA verification protocol in Algorithm 1 is $(t, q_v, 1/(p-1))$-sound against collusion and adaptive chosen message attacks, we will show that the adversary can only prove an invalid signature as valid with at most probability $1/(p-1)$. The challenger and the adversary play the following game.

(i) *Setup*. The challenger generates the system parameters $(\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e)$, chooses a random element $x \in \mathbb{Z}_p$, and sets $sk = x$ and $pk = (\vec{V}, PK)$ where $\vec{V}$ is a vector consisting of $n+1$ elements $V_0, V_1, \ldots, V_n$ randomly selected in $\mathbb{G}_1$ and $PK = e(g,g)^x$. Then it also computes $K_1 = e(g,g)$. Finally the challenger sends $(\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e, sk, pk)$ to the adversary.

(ii) *Queries*. The adversary $\mathscr{A}$ is allowed to make at most q_v server-aided verification queries. The challenger $\mathscr{C}$ responds by executing SA-Verify with the adversary $\mathscr{A}$, where the adversary $\mathscr{A}$ acts as the server and the challenger $\mathscr{C}$ acts as the verifier. At the end of each execution, the challenger returns the output of SA-Verify to the adversary $\mathscr{A}$.

(iii) *Output*. Eventually, the adversary $\mathscr{A}$ outputs a message m^*. The challenger $\mathscr{C}$ chooses a random element $\sigma^* = (\sigma_1, \sigma_2)$ in $\mathbb{G}_1^2 \setminus \Omega_{m^*}$, where Ω and Ω_{m^*} are, respectively, the signature space and the set of valid signatures of m^*. Then they interact with each other as described in the SA-Verify protocol. Concretely, the challenger chooses two random elements $r^*, d^* \in \mathbb{Z}_p^*$, computes $M = V_0\prod_{i\in\mathscr{M}} V_i$, $M' = M \cdot g^{r^*}$ and $\sigma_1' = \sigma_1 \cdot g^{d^*} \cdot \sigma_2^{r^*}$, and sends $(M', \sigma_1', \sigma_2)$ to the adversary. Then the adversary returns K_2^* and K_3^* to the challenger.

In the following, we will show that $K_2^* = K_3^* \cdot PK \cdot K_1^{d^*}$ happens with probability $1/(p-1)$.

The challenger sends $(M', \sigma_1', \sigma_2)$ to $\mathscr{A}$ such that

$$\begin{aligned} M' &= M \cdot g^{r^*}, \\ \sigma_1' &= \sigma_1 \cdot g^{d^*} \cdot \sigma_2^{r^*} \end{aligned} \tag{4}$$

$$\Downarrow$$

$$\begin{aligned} DL_g M' &= DL_g M + r^*, \\ DL_g \sigma_1' &= DL_g \sigma_1 + d^* + DL_g \sigma_2 \cdot r^*. \end{aligned} \tag{5}$$

Since the adversary can only obtain $(M', \sigma_1', \sigma_2)$ from the challenger, from the equation set (5), we can see that the adversary can only obtain d^* with probability $1/(p-1)$. Furthermore, from (6) below, we can directly deduce (7) as follows:

$$K_2^* = K_3^* \cdot PK \cdot K_1^{d^*} \tag{6}$$

$$\Downarrow$$

$$DL_{e(g,g)}\left(\frac{K_2^*}{K_3^*}\right) = DL_{e(g,g)} PK + d^*. \tag{7}$$

Since the adversary can only guess d^* with probability $1/(p-1)$, and (K_2^*/K_3^*) is uniquely determined by d^*, the adversary can only give out a pair (K_2^*, K_3^*) satisfying (6) with probabitity $1/(p-1)$. This completes the proof of Theorem 4. □

Theorem 5. *The SA verification protocol in Algorithm 1 satisfies $(t, q_v, 1/2)$-message privacy against collusion and adaptive chosen message attacks.*

Proof. In order to prove that the SA verification protocol in Algorithm 1 satisfies $(t, q_v, 1/2)$-message privacy against collusion and adaptive chosen message attacks, we will show that the adversary can only succeed with at most the probability $1/2$ in the game with the challenger described as follows.

(i) *Setup*. The challenger generates the system parameter $(\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e)$, and the secret key $sk = x$ and public key $pk = (\vec{V}, PK)$, where $\vec{V}$ is a vector consisting of $n+1$ elements $V_0, V_1, \ldots, V_n$ randomly selected in $\mathbb{G}_1$ and $PK = e(g,g)^x$. Then it also computes $K_1 = e(g,g)$. Finally the challenger sends $(\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e, sk, pk)$ to the adversary.

(ii) *Queries*. The adversary $\mathscr{A}$ is allowed to make at most q_v server-aided verification queries. The challenger $\mathscr{C}$ responds by executing SA-Verify with the adversary $\mathscr{A}$, and, at the end of each execution, returns the output of SA-Verify to the adversary $\mathscr{A}$.

(iii) *Challenge*. $\mathscr{A}$ outputs two messages m_0 and m_1 and sends them to the challenger $\mathscr{C}$. $\mathscr{C}$ chooses a bit $b \in \{0, 1\}$ at random and also chooses $\sigma = (\sigma_1, \sigma_2)$ either randomly from Ω_{m_0} or randomly from Ω_{m_1}, where Ω_{m_0} and Ω_{m_1} are, respectively, the signature spaces of m_0 and m_1. Then $\mathscr{C}$ and $\mathscr{A}$ interact with each other by running SA-Verify$(\mathscr{A}, \mathscr{C}^{(m_b,\sigma,pk,\mathsf{VString})})$, where $\mathscr{A}$ plays as a server and $\mathscr{C}$ plays as a verifier. Concretely, the challenger chooses two random elements $r^*, d^* \in \mathbb{Z}_p^*$, computes $M = V_0 \prod_{i \in \mathscr{M}} V_i$, $M' = M \cdot g^{r^*}$, and $\sigma_1' = \sigma_1 \cdot g^{d^*} \cdot \sigma_2^{r^*}$, and sends $(M', \sigma_1', \sigma_2)$ to the adversary. Then the adversary returns K_2^* and K_3^* to the challenger. After the interaction, $\mathscr{C}$ sends the output of SA-Verify$(\mathscr{A}, \mathscr{C}^{(m_b,\sigma,pk,\mathsf{VString})})$ to $\mathscr{A}$.

(iv) *Output*. Finally, $\mathscr{A}$ outputs a bit $b' \in \{0, 1\}$. We will show that $\mathscr{A}$ can only succeed with probability $1/2$.

From the equation set (5) in the proof of Theorem 4, we can see that there exist two pairs (M_0, r_0^*) and (M_1, r_1^*) satisfying $M' = M_b \cdot g^{r_b^*}$ for $b = 0, 1$, and for each r_b^*, there exist $p - 1$ pairs (σ_1, d^*) satisfying $\sigma_1' = \sigma_1 \cdot g^{d^*} \cdot \sigma_2^{r_b^*}$ for $b = 0, 1$. So the adversary cannot obtain anything about (M, σ_1). Though the adversary learns σ_2 which may correspond to m_0 or m_1, it only guess b correctly with probability $1/2$. This completes the proof of Theorem 4. □

Efficiency Analysis. The SA verification signature with privacy based on Waters signature above is computation saving and efficient. In the following, we will analyze the efficiency of SA-Verify algorithm by comparing that of Waters signature scheme. In Waters signature scheme [21], to verify a message-signature pair $(m, \sigma = (\sigma_1, \sigma_2))$, the verifier needs to compute $M = V_0 \prod_{i \in \mathscr{M}} V_i$ which takes n multiplications in $\mathbb{G}_1$, $PK \cdot e(m, \sigma_2)$ which takes 1 multiplication in $\mathbb{G}_T$, and two pairings $e(m, \sigma_2)$ and $e(\sigma_1, g)$. However, in our server-aided verification signature scheme, the verifier can first precompute a pairing $e(g, g)$ which can be used by multiple SA-Verify protocols. Then in a SA-Verify protocol, we can see that the verifier needs to compute totally 3 exponentiations in $\mathbb{G}_1$ and 1 exponentiation in $\mathbb{G}_T$ as well as $3 + n$ multiplications in $\mathbb{G}_1$ and 3 multiplications in $\mathbb{G}_T$. As we know that, performing a pairing on an elliptic curve requires much more computational cost than executing both an exponentiation and a multiplication. So our SA verification signature scheme based on Waters signature is computation saving and efficient.

The concrete computation cost comparison of the verifier in the verification of Waters Signature [21] and our SA-Verify in Algorithm 1 is shown in Table 1.

Table 1: Efficiency of the SA verification signature scheme.

Verification	Pairing	Exponentiation	Multiplication
Waters [21]	2	0	$n(\mathbb{G}_1)$
Our Scheme	0	$3(\mathbb{G}_1) + 1(\mathbb{G}_T)$	$(3 + n)(\mathbb{G}_1) + 3(\mathbb{G}_T)$

4. Server-Aided Verification Signature with Message-Signature Privacy

In this section, we will present a stronger definition of privacy, namely, message-signature privacy against collusion and adaptive chosen message attacks. Then based on BLS signature scheme [18], a concrete server-aided verification scheme with this privacy property will be presented. We assume that the server can obtain the key pair (pk, sk) of the signer and cannot obtain the message-signature pairs that have been created by the signer. Under this assumption, the server cannot obtain anything about the message-signature pair.

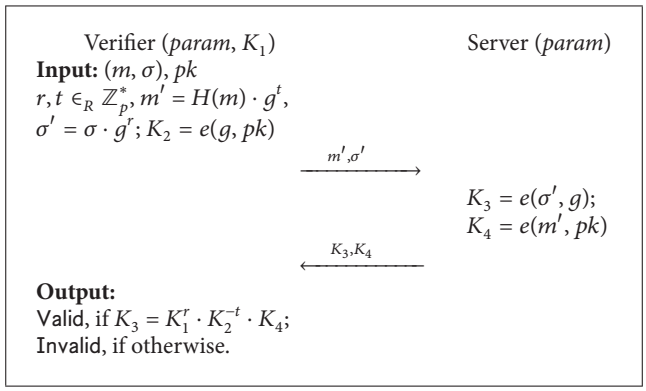
4.1. Definition of Message-Signature Privacy. A server-aided verification signature scheme with message-signature privacy SAV-Σ also consists of six algorithms: ParamGen, KeyGen, Sign, Verify, SA-Verifier-Setup, and SA-Verify.

Definition 6 (message-signature privacy of SA-Verify). We say that SA-Verify satisfies (t, q_v, ε)-message-signature privacy against collusion and adaptive chosen message attacks if there exists no adversary $\mathscr{A}$ who runs in time at most t, makes at most q_v server-aided verification queries, and succeeds with probability at least ε in the following game with the challenger $\mathscr{C}$. The game is defined as follows.

(i) *Setup*. The challenger $\mathscr{C}$ runs the algorithms ParamGen, KeyGen and SA-Verifier-Setup to obtain the system parameter *param*, one key pair (sk, pk), and VString. The adversary $\mathscr{A}$ is given *param* and (sk, pk). Similar to the definition of message privacy for SA verification signature, $\mathscr{A}$ can generate any message-signature pair with the key pair (sk, pk); however as we assumed, it cannot obtain any message-signature pair that has been created by the signer before.

(ii) *Queries*. Proceeding adaptively, the adversary $\mathscr{A}$ is allowed to make at most q_v server-aided verification queries. The challenger $\mathscr{C}$ responds by executing SA-Verify with the adversary $\mathscr{A}$, where the adversary $\mathscr{A}$ acts as the server and the challenger $\mathscr{C}$ acts as the verifier. At the end of each execution, the challenger returns the output of SA-Verify to the adversary $\mathscr{A}$.

(iii) *Challenge*. $\mathscr{A}$ outputs two pairs (M_0, σ_0) and (M_1, σ_1), where σ_i is a valid signature on M_i for $i = 1, 2$. Then $\mathscr{A}$ sends them to the challenger $\mathscr{C}$. $\mathscr{C}$ chooses a bit $b \in \{0, 1\}$ at random and interacts with $\mathscr{A}$ by running SA-Verify$(\mathscr{A}, \mathscr{C}^{(M_b,\sigma_b,pk,\mathsf{VString})})$ where $\mathscr{A}$ plays as a server and $\mathscr{C}$ plays as a verifier. After the interaction, $\mathscr{C}$ sends the output of SA-Verify$(\mathscr{A}, \mathscr{C}^{(M_b,\sigma_b,pk,\mathsf{VString})})$ to $\mathscr{A}$.



Algorithm 2: SA verification Signature with message-signature privacy based on BLS signature.

(iv) *Output*. Finally, $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$. We say that $\mathcal{A}$ wins the game with probability ε if

$$\Pr\left[b = b'\right] \geq \varepsilon. \tag{8}$$

4.2. Concrete SA Verification Signature with Message-Signature Privacy. In this section, we will present a SA verification signature scheme which satisfies message-signature privacy against collusion and adaptive chosen message attacks. This scheme is constructed based on BLS signature [18], which also consists of six algorithms: ParamGen, KeyGen, Sign, Verify, SA-Verifier-Setup, and SA-Verify. The first four algorithms are the same as those in BLS signature scheme [18]. By executing the SA-Verifier-Setup and SA-Verify algorithms, the computational load of the verifier can be reduced. In BLS signature [18], the verifier has to compute two pairings; however in the following SAV-Σ, it needs only to compute a pairing.

(i) $param \leftarrow$ ParamGen. Let κ be a security parameter, $(\mathbb{G}_1, \mathbb{G}_T)$ be bilinear groups, where $|\mathbb{G}_1| = |\mathbb{G}_T| = p$ for some prime number $p \geq 2^\kappa$, and g be a generator of $\mathbb{G}_1$. $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ is a bilinear mapping. $H : \{0, 1\}^* \rightarrow \mathbb{G}_1$ is a hash function. The system parameter $param = (\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e, H)$.

(ii) $(pk, sk) \leftarrow$ KeyGen($param$). Given the system parameters $(\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e)$, the signer chooses a random element $x \in \mathbb{Z}_p$, and sets the public key $pk = g^x$ and the secret key $sk = x$.

(iii) $\sigma \leftarrow$ Sign($param, m, sk, pk$). For a message, the signer generates the signature $\sigma = H(m)^x$.

(iv) {Valid, Invalid} $\leftarrow$ Verify($param, m, \sigma, pk$). The verifier takes as inputs a claimed message-signature pair (m, σ), and outputs Valid if and only if $e(\sigma, g) = e(H(m), pk)$, and otherwise outputs Invalid.

(v) VString $\leftarrow$ SA-Verify-Setup($param$). The verifier takes as inputs the system parameter $(\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e)$ and computes $K_1 = e(g, g)$ as VString.

(vi) {Valid, Invalid} $\leftarrow$ SA-Verify(Server$^{(param)}$, and Verifier$^{(m,\sigma,pk,\text{VString})}$). This is an interactive protocol between the server and the verifier which is shown in Algorithm 2.

(1) Verifier, for a message-signature pair (m, σ), blinds the pair by selecting randomly $r, t \in \mathbb{Z}_p^*$ and computing $m' = H(m) \cdot g^t$, $\sigma' = \sigma \cdot g^r$; $K_2 = e(g, pk)$, and sends (m', σ') to the verifier.

(2) Server computes $K_3 = e(\sigma', g)$, $K_4 = e(m', pk)$ and returns K_3, K_4 to the verifier.

(3) Verifier checks the equation $K_3 = K_1^r \cdot K_2^{-t} \cdot K_4$, and outputs Valid if it holds, and otherwise outputs Invalid.

Correctness of SA-Verify. For a claimed message-signature pair (m, σ), when the verifier and the server are both honest, namely, the verifier correctly computes m', σ' and K_2, and the server correctly computes K_3 and K_4; then the verification equation $K_3 = K_1^r \cdot K_2^{-t} \cdot K_4$ holds if the message-signature pair (m, σ) is valid and otherwise does not hold. Consider

$$\begin{aligned} K_3 &= e\left(\sigma', g\right) \\ &= e\left(\sigma, g\right) \cdot e\left(g^r, g\right) \\ &= e\left(H\left(m\right)^x, g\right) \cdot e\left(g, g\right)^r \\ &= e\left(H\left(m\right), pk\right) \cdot K_1^r \\ &= e\left(m' \cdot g^{-t}, pk\right) \cdot K_1^r \\ &= e\left(m', pk\right) \cdot e\left(g, pk\right)^{-t} \cdot K_1^r \\ &= K_4 \cdot K_2^{-t} \cdot K_1^r. \end{aligned} \tag{9}$$

In the following, by Theorems 7 and 8, we will show that our SA verification signature scheme above is secure under our security model; namely, the SA verification protocol described in Algorithm 2 is sound against collusion and adaptive chosen message attacks and also satisfies message-signature privacy against collusion and adaptive chosen message attacks.

Theorem 7. *The SA verification protocol described in Algorithm 2 satisfies soundness against collusion and adaptive chosen message attacks.*

Proof. In order to prove that the SA verification protocol in Algorithm 2 is $(t, q_v, 1/(p-1))$-sound against collusion and adaptive chosen message attacks, we will show that the adversary can only prove an invalid signature as valid with at most probability $1/(p-1)$. The challenger and the adversary play the following game.

(i) *Setup*. The challenger generates the system parameter $(\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e, H)$, chooses a random element $x \in \mathbb{Z}_p$, and sets $sk = x$ and $pk = g^x$. Then it also computes $K_1 = e(g, g)$. Finally the challenger sends $(\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e, H, sk, pk)$ to the adversary.

(ii) *Queries*. The adversary $\mathcal{A}$ is allowed to make at most q_v server-aided verification queries. The challenger $\mathcal{C}$ responds by executing SA-Verify with the adversary $\mathcal{A}$. At the end of each execution, the challenger returns the output of SA-Verify to the adversary.

(iii) *Output*. Eventually, the adversary $\mathscr{A}$ outputs a message m^*. The challenger $\mathscr{C}$ chooses a random element σ^* in $\mathbb{G}_1 \backslash \Omega_{m^*}$, where Ω_{m^*} is the set of valid signatures of m^*. Then they interact with each other as described in the SA-Verify protocol. Concretely, the challenger chooses two random elements $r^*, t^* \in \mathbb{Z}_p^*$, computes $m' = H(m) \cdot g^{t^*}$, $\sigma' = \sigma \cdot g^{r^*}$ and $K_2 = e(g, pk)$, and sends (m', σ') to the adversary. Then the adversary returns K_3^* and K_4^* to the challenger.

In the following, we will show that $K_3^* = K_1^{r^*} \cdot K_2^{-t^*} \cdot K_4^*$ happens with probability $1/(p-1)$.

From (10), we can directly deduce (11):

$$\begin{aligned} m' &= H(m) \cdot g^{t^*}, \\ \sigma' &= \sigma \cdot g^{r^*} \\ &\Downarrow \\ DL_g m' &= DL_g H(m) + t^*, \\ DL_g \sigma' &= DL_g \sigma + r^*. \end{aligned} \tag{10, 11}$$

Since (t^*, r^*) is chosen randomly from $\mathbb{Z}_p^*$, from the equation set (7), we can see that the adversary can only obtain m and σ with probability $1/(p-1)$. Furthermore, from the following, (13) can be deduced:

$$\begin{aligned} K_3^* &= K_1^{r^*} \cdot K_2^{-t^*} \cdot K_4^* \\ &\Downarrow \\ DL_{e(g,g)}\left(\frac{K_3^*}{K_4^*}\right) &= DL_{e(g,g)} K_1^* \cdot r^* - DL_{e(g,g)} K_2^* \cdot t^*. \end{aligned} \tag{12, 13}$$

We can see that K_3^*/K_4^* is determined by r^* and t^*. Since $DL_{e(g,g)} K_1^* \cdot r^* - DL_{e(g,g)} K_2^* \cdot t^*$ is a random element in $\mathbb{Z}_p^*$, the adversary can only give out a pair (K_3^*, K_4^*) satisfying (11) with probabitity $1/(p-1)$. This completes the proof of Theorem 7. □

Theorem 8. *The SA verification protocol described in Algorithm 2 satisfies message-signature privacy against collusion and adaptive chosen message attacks.*

Proof. In order to prove that the SA verification protocol in Algorithm 2 satisfies $(t, q_v, 1/2)$-message-signature privacy against collusion and adaptive chosen message attacks, we will show that the adversary can only succeed with at most probability $1/2$ in the game with the challenger described as follows.

(i) *Setup*. The challenger generates the system parameter $(\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e, H)$, and the secret key $sk = x$ and the public key $pk = g^x$. Then it also computes $K_1 = e(g, g)$. Finally the challenger sends $(\kappa, \mathbb{G}_1, \mathbb{G}_T, p, g, e, H, sk, pk)$ to the adversary.

Table 2: Efficiency of the SA verification signature scheme.

Verification	Pairing	Exponentiation	Multiplication	Hash
BLS [18]	2	0	0	1
Our scheme	1	$2(\mathbb{G}_1) + 2(\mathbb{G}_T)$	$2(\mathbb{G}_1) + 2(\mathbb{G}_T)$	1

(ii) *Queries*. The adversary $\mathscr{A}$ is allowed to make at most q_v server-aided verification queries. The challenger $\mathscr{C}$ responds by executing SA-Verify with the adversary $\mathscr{A}$, and, at the end of each execution, returns the output of SA-Verify to the adversary $\mathscr{A}$.

(iii) *Challenge*. $\mathscr{A}$ outputs two pairs (m_0, σ_0) and (m_1, σ_1), where σ_i is a valid signature on m_i for $i = 1, 2$. Then it sends them to the challenger $\mathscr{C}$. $\mathscr{C}$ chooses a bit $b \in \{0, 1\}$ at random and interacts with $\mathscr{A}$ by running SA-Verify$(\mathscr{A}, \mathscr{C}^{m_b, \sigma_b, pk, \mathsf{VString}})$. Concretely, the challenger chooses two random elements $r^*, t^* \in \mathbb{Z}_p^*$, computes $m_b' = H(m_b) \cdot g^{t^*}$, $\sigma_b' = \sigma_b \cdot g^{r^*}$ and $K_2 = e(g, pk)$, and sends (m_b', σ_b') to the adversary. Then the adversary returns K_3^* and K_4^* to the challenger. After the interaction, $\mathscr{C}$ sends the output of SA-Verify$(\mathscr{A}, \mathscr{C}^{(m_b, \sigma_b, pk, \mathsf{VString})})$ to $\mathscr{A}$.

(iv) *Output*. Finally, $\mathscr{A}$ outputs a bit $b' \in \{0, 1\}$. We will show that $\mathscr{A}$ can only succeed with probability $1/2$.

From the equation set (10) in the proof of Theorem 4, we can see that there exist two pairs (m_0, t_0^*) and (m_1, t_1^*) satisfying $m' = H(m_b) \cdot g^{t_b^*}$ for $c = 0, 1$, and there exist two pairs (σ_0, r_0^*) and (σ_1, r_1^*) satisfying $\sigma' = \sigma_b \cdot g^{r^*}$ for $c = 0, 1$. So the adversary cannot obtain anything about (m_b, σ_b), and it only guesses b correctly with probability $1/2$. This completes the proof of Theorem 8. □

Efficiency Analysis. In the following, we will show that our SA verification signature scheme based on BLS signature above is computation saving and efficient. We will analyze the efficiency of SA-Verify algorithm by comparing that of BLS Signature scheme [18]. In BLS signature scheme [18], to verify a message-signature pair (m, σ), the verifier needs to compute $H(m)$ which takes 1 hash function and two bilinear pairings $e(H(m), pk)$ and $e(\sigma, g)$. However, in our server-aided verification signature scheme, the verifier can first precompute a pairing $e(g, g)$ which can be used by multiple SA-Verify protocols. Then in a SA-Verify protocol, the verifier needs to compute totally 2 exponentiations in $\mathbb{G}_1$ and 2 exponentiation in $\mathbb{G}_T$ as well as 2 multiplications in $\mathbb{G}_1$ and 2 multiplications in $\mathbb{G}_T$ and 1 hash function. From the comparison, we can see that our SA verification signature scheme based on BLS signature is computation saving. The concrete computation cost comparison of the verifier in the verification of BLS signature and SA-Verify in Algorithm 2 is shown in Table 2.

5. Conclusion

In this paper, we studied the SA verification signature schemes with message-signature privacy for mobile computing. A power-constrained mobile device can outsource the verification of a signature to a server with powerful resources in order to reduce its computational load. We first present two definitions for privacy of server-aided verification protocol, respectively, named message privacy and message-signature privacy under collusion and adaptive chosen message attacks. Then under our security models, two concrete constructions based on existing signature schemes were presented and proved secure. By efficiency analysis, we showed that the two concrete schemes are both computation saving and efficient.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (nos. 61202466, U1135004, 61170080, and 61100224), 973 Program under Grant no. 2014CB360501, Foundation for Distinguished Young Talents in Higher Education of Guangdong, China (no. 2012LYM_0017), and Fundamental Research Funds for the Central Universities (South China University of Technology) (no. 2014ZM0032). An abstract of this paper has been presented in the 4th International Conference on Emerging Intelligent Data and Web Technologies (2013), 414–421 [31].

References

[1] A. Durresi and M. Denko, "Preface: advances in mobile communications and computing," *Mobile Information Systems*, vol. 5, no. 2, pp. 101–103, 2009.

[2] A. Durresi and M. Denko, "Advances in wireless networks," *Mobile Information Systems*, vol. 5, no. 1, pp. 1–3, 2009.

[3] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *Advances in Cryptology—CRYPTO 2011*, vol. 6841 of *Lecture Notes in Computer Science*, pp. 111–131, Springer, Berlin, Germany, 2011.

[4] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: outsourcing computation to untrusted workers," in *Advances in Cryptology—CRYPTO 2010*, vol. 6223 of *Lecture Notes in Computer Science*, pp. 465–482, Springer, Berlin, Germany, 2010.

[5] K. M. Chung, Y. Kalai, and S. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *Advances in Cryptology—CRYPTO 2010*, vol. 6223 of *Lecture Notes in Computer Science*, pp. 483–501, Springer, Berlin, Germany, 2010.

[6] D. Fiore and R. Gennaro, "Publicly verifiable delegation of large polynomials and matrix computations, with applications," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS '12)*, pp. 501–512, ACM, October 2012.

[7] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: verifiable computation from attribute-based encryption," in *Theory of Cryptography*, vol. 7194 of *Lecture Notes in Computer Science*, pp. 422–439, Springer, Berlin, Germany, 2012.

[8] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Optimal verification of operations on dynamic sets," in *Advances in Cryptology—CRYPTO 2011*, vol. 6841 of *Lecture Notes in Computer Science*, pp. 91–110, 2011.

[9] P. Béguin and J.-J. Quisquater, "Fast server-aided RSA signatures secure against active attacks," in *Advances in Cryptology—CRYPT0 1995*, vol. 963 of *Lecture Notes in Computer Science*, pp. 57–69, Springer, Berlin, Germany, 1995.

[10] J.-J. Quisquater and M. de Soete, "Speeding up smart card RSA computation with insecure coprocessors," in *Proceedings of the Smart Cards 2000*, pp. 191–197, 1989.

[11] C. H. Lim and P. J. Lee, "Security and performance of server-aided RSA computation protocols," in *Advances in Cryptology—CRYPTO'1995*, vol. 963 of *Lecture Notes in Computer Science*, pp. 70–83, Springer, Berlin, Germany, 1995.

[12] P. Nguyen and J. Stern, "The Béguin-Quisquater server-aided RSA protocol from Crypto '95 is not secure," in *Advances in Cryptology—ASIACRYPT 1998*, vol. 1514 of *Lecture Notes in Computer Science*, pp. 372–379, Springer, Berlin, Germany, 1998.

[13] M. Girault and J. J. Quisquater, "GQ + GPS = new ideas + new protocols," in *Proceedings of the Eurocrypt 2002-Rump Session*, April-May 2002.

[14] M. Girault and D. Lefranc, "Server-aided verification: theory and practice," in *Advances in Cryptology—ASIACRYPT 2005*, vol. 3788 of *Lecture Notes in Computer Science*, pp. 605–623, Springer, Berlin, Germany, 2005.

[15] W. Wu, Y. Mu, W. Susilo, and X. Y. Huang, "Provably secure server-aided verification signatures," *Computers and Mathematics with Applications*, vol. 61, no. 7, pp. 1705–1723, 2011.

[16] G. C. C. F. Pereira, M. A. Simplicío Jr., M. Naehrig, and P. S. L. M. Barreto, "A family of implementation-friendly BN elliptic curves," *Journal of Systems and Software*, vol. 84, no. 8, pp. 1319–1326, 2011.

[17] S. Hohenberger and A. Lysyanskaya, "How to securely outsource cryptographic computations," in *Theory of Cryptography*, vol. 3378 of *Lecture Notes in Computer Science*, pp. 264–282, Springer, Berlin, Germany, 2005.

[18] D. Boneh, G. Lynn, and H. Shacham, "Short signature from the Weil pairing," in *Advances in Cryptology—Asiacrypt 2001*, vol. 2248 of *Lecture Notes in Computer Science*, pp. 514–532, Springer, Berlin, Germany, 2001.

[19] D. Boneh and X. Boyen, "Short signatures without random oracles," in *Advances in Cryptology—EUROCRYPT 2004*, vol. 3027 of *Lecture Notes in Computer Science*, pp. 382–400, Springer, Berlin, Germany, 2004.

[20] F. Zhang, R. Safavi-Naini, and W. Susilo, "An efficient signature scheme from bilinear pairing and its applications," in *Public Key Cryptography—PKC 2004*, vol. 2947 of *Lecture Notes in Computer Science*, pp. 277–290, 2004.

[21] B. Waters, "Efficient identity-based encryption without random oracles," in *Advances in Cryptology—EUROCRYPT 2005*, vol. 3494 of *Lecture Notes in Computer Science*, pp. 114–127, 2005.

[22] X. Chen, F. Zhang, Y. Mu, and W. Susilo, "Efficient provably secure restrictive partially blind signatures from bilinear pairings," in *Financial Cryptography and Data Security*, vol. 4107 of *Lecture Notes in Computer Science*, pp. 251–265, Springer, Berlin, Germany, 2006.

[23] X. Chen, F. Zhang, W. Susilo, H. Tian, J. Li, and K. Kim, "Identity-based chameleon hash scheme without key exposure," in *Information Security and Privacy*, vol. 6168 of *Lecture Notes in Computer Science*, pp. 200–215, Springer, 2010.

[24] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, 1988.

[25] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology*, pp. 199–203, Plenum, 1983.

[26] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López, "Faster explicit formulas for computing pairings over ordinary curves," in *Advances in Cryptology—EUROCRYPT 2011*, vol. 6632 of *Lecture Notes in Computer Science*, pp. 48–68, Springer, 2011.

[27] J.-L. Beuchat, J. E. Gonzalez-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya, "High-speed software implementation of the optimal ate pairing over Barreto-Naehrig curves," in *Pairing-Based Cryptography—Pairing 2010*, vol. 6487 of *Lecture Notes in Computer Science*, pp. 21–39, Springer, 2010.

[28] D. Hankerson, A. J. Menezes, and M. Scott, "Software implementation of pairings," in *Identity-Based Cryptography*, M. Joye and G. Neven, Eds., pp. 188–206, IOS Press, 2008.

[29] M. Naehrig, R. Niederhagen, and P. Schwabe, "New software speed records for cryptographic pairings," in *Progress in Cryptology—LATINCRYPT 2010*, vol. 6212 of *Lecture Notes in Computer Science*, pp. 109–123, Springer, Berlin, Germany, 2010.

[30] C. Costello, K. Lauter, and M. Naehrig, "Attractive subfamilies of BLS curves for implementing high-security pairings," in *Progress in Cryptology—INDOCRYPT 2011*, vol. 7107 of *Lecture Notes in Computer Science*, pp. 320–342, Springer, 2011.

[31] L. Xu and S. Tang, "Server-aided verification signatures with privacy," in *Proceedings of the 4th International Conference on Emerging Intelligent Data and Web Technologies (EIDWT '13)*, pp. 414–421, September 2013.