A server-aided verification signature scheme without random oracles

Bin Wang and Qing Zhao

Information Engineering College of Yangzhou University

No.196 West HuaYang Road, Yangzhou City, Jiangsu Province, P.R.China

E-mail: jxbin76@yahoo.cn

Abstract: Server-aided verification(SAV) signature is useful for power-constrained devices since a powerful server can assist in performing costly operations such as pairing operations. Wu et al. [13] defined three security notions for SAV protocol to prevent a server from convincing a verifier that an invalid signature is valid. Security against strong collusion attack provides the strongest security guarantee among these notions. They [13] constructed SAV protocols that meet the requirement of these notions respectively. But they did not provide concrete running time to show that the running time of a verifier in their SAV protocol is strictly less than that of a verifier in the original verification protocol. In addition, a problem left open by their work is to design SAV signature which is unforgeable without random oracles as well as sound against strong collusion attack. To address the above issues, we first choose to design a SAV protocol called SAV-Hofheinz for a short signature proposed by Hofheinz unforgeable in the standard model. Then we implement SAV-Hofheinz by the PBC library and shows that the running time of a verifier in SAV-Hofheinz is strictly less than that of a verifier in the verification protocol of Hofheinz short signature.

Keywords: Server-aided verification; Short signature; Pairing; Power-constrained devices; The PBC library

1. Introduction

With the development of power-constrained devices such as smart cards and mobile terminals, it is desirable to design cryptographic protocols with computational cost suitable for such kind of devices. Due to its elegant algebraic properties, bilinear pairings have been used to construct various cryptographic schemes. For example, BLS signature scheme [2] that uses bilinear pairings has the shortest signature size among provable secure signature schemes.

Generally speaking, pairing-based signature schemes allow a signer to compute a signature in a relatively fast way by computing some multiplication or exponentiation operations over groups. However, a verifier has to compute several pairing operations to verify a signature. Currently, as pairing requires much more computational cost than exponentiation, it is a challenging task to speed up verification steps in pairing-based signature schemes when applied to power-constrained devices.

A solution to this problem is referred to as "server-aided verification". Informally speaking, a server-aided verification signature scheme $\mathrm{SAV}-\Sigma$ consists of a standard signature scheme Σ and a server-aided verification protocol. In a client-server environment, a power-constrained device (client) is connected to a powerful server who can assist the client to perform costly operations involved in the original verification algorithm of Σ. As the server may be untrusted, we must prevent a malicious server from convincing a verifier with a limited computational power that an invalid signature is valid.

Server-aided verification protocol was introduced first by Quisquater and De Soete to speed up RSA verification with a small exponent [3]. Then Lim and Lee gave server-aided computation protocols based on the ”randomization” of the verification equation [8]. However, the verifier must perform heavy pre-computation before carrying out the server-aided protocol. Girault and Quisquater [5] presented an approach for server-aided verification protocols which does not require pre-computation or randomization and is computationally secure based on the hardness of a sub-problem of the underlying complexity problem in the original signature scheme.

Hohenberger and Lysyanskaya addressed the situation in which the server is made of two untrusted softwares, which are assumed not to communicate with each other [7]. This assumption is strong but it allows a very light public computation task. Girault and Lefrance [4] provided a security model for server-aided verification protocols without this strong assumption. They proposed a generic method for designing SAV versions of schemes based on bilinear maps, which can be applied to the Boneh-Boyen signature schemes [1] and the Zhang-Safavi-Naini-Susilo [14] signature scheme.

Recently, Wu et al. [12,13] defined three security notions for server-aided verification signatures, i.e., existential unforgeability, security against collusion attack and security against strong collusion attack to extend the security model for SAV signature defined in [4]. Their definition of existential unforgeability for SAV signature combines standard existential unforgeability with soundness notion in server-aided verification protocol, which requires a computationally bounded malicious server, who is unable to corrupt the original signer, is not able to convince a verifier that an invalid signature is valid. Security against collusion attack allows collusion between a malicious server and the original signer, but the server has no control over the choice of invalid signature for the challenge message. Security against strong collusion attack provides the strongest security guarantee by allowing a malicious server to choose a invalid signature for the challenge message.

Then Wu et al. [13] designed server-aided verification protocols for waters signature [11], BLS signature [2] respectively, which satisfy both existential unforgeability for SAV signature and security against collusion attack. However, only a server-aided verification protocol for BLS signature was presented to meet the requirement of security against strong collusion attack. As the unforgeability of BLS signature is proven in the random oracle model, the work of [13] leaves two points to be desired:

(1) As security proofs in the random oracle model rely on the idealized assumption that cryptographic hash functions can be modeled by random functions, it is desirable to obtain a server aided verification protocol that meets the requirement of security against strong collusion attack for a signature scheme unforgeable without random oracles.

(2) The key idea of [13] is to use less expensive exponentiation operations to replace one pairing operation. Computational cost is measured by the number of operations performed by a verifier in verification protocol [13]. However, it is important to estimate the concrete running time of a verifier in order to present convincing evidence that the computational cost of a verifier in server-aided verification protocol is indeed strictly less than that of a verifier in the original verification protocol when running on power-constrained devices.

To address the above two issues, we first choose to design a server-aided verification protocol called SAV-Hofheinz for a short signature proposed by Hofheinz unforgeable in the standard model [6] with performance almost comparable to that of BLS short signature. When designing the server-aided verification protocol for Hofheinz short signature, we observe that the online computation can be speeded up by doing some exponentiation operations in advance.

Second, we implement SAV-Hofheinz protocol with the PBC library [10]. The result shows that the running time of a verifier in SAV-Hofheinz protocol is strictly less than that of a verifier in the original verification protocol when running on our testbed.

The rest of this paper is organized as follows. At first, we review the syntax of server-aided verification signature in section 3. Then we describe security against collusion attack and security against strong collusion attack defined in [13]. In the following, we point out that the server-aided verification protocol for waters signature which satisfies the requirement of security against collusion attack [13] is not secure against strong collusion attack. So we design a server-aided verification protocol for the short signature scheme [6] and prove that our server-aided verification protocol meets the requirement of security against strong collusion attack defined in [13]. Finally, we implement the proposed server-aided verification protocol with the PBC library and compare the running time of a verifier in our server-aided verification protocol with that of a verifier in the original verification protocol.

2. Bilinear pairing

Given a security parameter k, an efficient algorithm $PG(1^k)$ outputs (e, G, G_T, g, p), where G is a cyclic group of a prime order p generated by g. G_T is a cyclic group of the same order, and let $e: G \times G \rightarrow G_T$ be a efficiently computable bilinear function with the following properties:

1. Bilinear: $e(g^a, g^b) = e(g, g)^{ab}$, for all $a, b \in Z_p$.

2. Non-degenerate: $e(g, g) \neq 1_{G_T}$

3. Syntax of server-aided verification signature

A server-aided verification signature $\mathrm{SAV}\text{-}\Sigma$ consists of a standard digital signature scheme $\Sigma = (\mathrm{Setup}, \mathrm{KG}, \mathrm{Sig}, \mathrm{Ver})$ and two additional algorithms SA-Ver-Setup, SA-Ver.

1. **Setup**(1^k): Takes as input a security parameter k, outputs the public system parameters

param, which defines message space $\mathbb{M}$ and the signature space $\mathbb{Q}$.

2. **KeyGen**(1^k): Takes **param** as input and outputs a secret/public key pair (sk, pk).

3. **Sig**(m, sk): Takes as input a signing key sk and a message m, then outputs a signature σ.

4. **Ver**(m, σ, pk): Takes as input a verification key pk, a message m and a purported signature σ, then outputs either 1 or 0 to denote “accept” or "reject".

5. **SA-Ver-Setup**($param$): A verifier takes **param** as input and computes a string $VString$ that will be used in the server aided verification protocol **SA-Ver**.

6. **SA-Ver**: $\langle \text{Server}(param), \text{Verifier}(m, \sigma, pk, VString)\rangle \rightarrow (\perp, b)$ is used to denote the interactive execution between Server and Verifier, where $param$ is the input of Server and $(m, \sigma, pk, VString)$ is the private input of Verifier who has limited computational ability and need check validity of the signature σ with the help of Server. The private output of Verifier is bit b. b =1 denotes that Verifier accept σ as valid by interacting with Server.

A distinct feature of $\text{SAV-}\Sigma$ is the property of computation-saving, which means that the computational cost of a verifier in $\text{SAV-}\Sigma$ should be strict less than that of a verifier in Σ.

Security of $\text{SAV-}\Sigma$ must include standard existential unforgeability of Σ and soundness of **SA-Ver,** which means a server cannot convince a honest verifier with a limited computational power that an invalid signature is valid. As we design $\text{SAV-}\Sigma$ based on some existing unforgeable signature schemes, the major concern of this paper is the soundness of **SA-Ver**.

4. Soundness of $\text{SAV-}\Sigma$

4.1 Soundness of $\text{SAV-}\Sigma$ against collusion and adaptive chosen message attack

By allowing a server to collude with the original signer, soundness of $\text{SAV-}\Sigma$ against collusion and adaptive chosen message attack is defined in experiment $Exp^{\text{SAV-}\Sigma}_{1^k, CCMA}(A)$. In the experiment, the adversary A who interacts with a challenger C has access to a oracle

$\text{O-SAV}(\cdot,\cdot)$. We adapt our definition from those given in [13].

$Exp^{\text{SAV-}\Sigma}_{1^k,CCMA}(A)$

$param \leftarrow \text{Setup}(1^k)$, $(pk,sk) \leftarrow \text{KeyGen}(1^k)$, $VString \leftarrow \text{SA-Ver-Setup}(param)$.

Then the challenger C sets $d \leftarrow 0$.

$m^* \leftarrow A^{\text{O-SAV}}(pk,sk)$.

The challenger picks $\sigma^* \leftarrow \mathbb{Q} \setminus Q_{m^*}$, where Q_{m^*} denotes the set of valid signatures with respect to the challenge message m^*.

Subsequently, the challenger plays the role of verifier to interact with A in **SA-Ver** protocol**:** $\langle \text{A}(param), \text{C}(m^*, \sigma^*, pk, VString)\rangle \rightarrow (\perp, b^*)$

If $b^* = 1$ then $d \leftarrow 1$

The formalization of the oracle O-SAV is defined as follows:

Oracle $\text{O-SAV}(m,\sigma)$

Runs an algorithm O who plays the role of verifier to interact with A in **SA-Ver** protocol: $\langle \text{A}(param), \text{O}(m, \sigma, pk, VString)\rangle \rightarrow (\perp, b)$

return b

The advantage of A is $\Pr[Exp^{\text{SAV-}\Sigma}_{1^k,CCMA}(A)] = 1$.

Definition 1: An adversary A is said to (t, q_v, ε) break soundness of SA-Ver in $\text{SAV-}\Sigma$ if A runs in time at most t, makes at most q_v calls to the oracle O-SAV, and succeeds with advantage ε in experiment $Exp^{\text{SAV-}\Sigma}_{1^k,CCMA}(A)$.

4.2 SAV-Waters scheme

Wu et al. constructed SAV-Waters scheme [13] with soundness against collusion and adaptive chosen message attack. The details of their scheme can be described as follows:

1. Setup: Given the security parameter k, $PG(1^k)$ outputs (e, G, G_T, g, p). Messages to be signed will be represented by n bits. The common parameter is $param = (e, G, G_T, g, p)$.

2. KeyGen: Picks a vector $\vec{\boldsymbol{V}} = (V_0, V_1 \cdots, V_n) \leftarrow_R G^{n+1}$ and $x \leftarrow_R Z_p$. The public key is $pk = (pk_1 = \vec{\boldsymbol{V}}, pk_2 = e(g, g)^x)$. The secret key sk is x.

3. Sig: Given a message $m \in \{0,1\}^n, m = m_1 \cdots m_n$, the signature is generated as follows: $\sigma = (\sigma_1, \sigma_2) = (g^x (V_0 \prod_{i=1}^{n} V_i^{m_i})^r, g^r)$, where $r \leftarrow_R Z_p$.

4. Ver: Given the public key $pk = (pk_1 = \vec{\boldsymbol{V}}, pk_2 = e(g, g)^x)$, a message m and the purported signature $\sigma = (\sigma_1, \sigma_2)$, outputs 1 if $e(\sigma_1, g) = pk_2 \cdot e(V_0 \prod_{i=1}^{n} V_i^{m_i}, \sigma_2)$.

5. SA-Ver-Setup($param$): A verifier takes **param** as input an computes $K_1 = e(g, g)$. $VString = K_1$.

6. SA-Ver: The verifier (resp., Server) takes $(m, \sigma, pk, VString)$ (resp., **param**) as input. Then the **SA-Ver** protocol is run as follows:

(1) The verifier parses $pk = (pk_1 = \vec{\boldsymbol{V}}, pk_2 = e(g, g)^x)$, $\sigma = (\sigma_1, \sigma_2)$, $VString = K_1$ and computes $\sigma_1^{/} = \sigma_1 \cdot g^d$, where $d \leftarrow_R Z_p$.

Verifier $\rightarrow$ Server: $m, \sigma_1^{/}, \sigma_2$

(2) The server computes $K_2 = e(\sigma_1^{/}, g)$, $K_3 = e(V_0 \prod_{i=1}^{n} V_i^{m_i}, \sigma_2)$.

Server $\rightarrow$ Verifier: K_2, K_3

(3) The verifier outputs 1 if $K_2 = pk_2 \cdot K_3 \cdot (K_1)^d$.

4.3 Soundness of SAV-Σ against strong collusion and adaptive chosen message attack

Soundness of $\text{SAV-}\Sigma$ against strong collusion and adaptive chosen message attack is defined in experiment $Exp_{1^k,SCCMA}^{\text{SAV-}\Sigma}(A)$. In the experiment, the adversary A is allowed to choose an invalid signature for the challenge message m^* by itself.

$Exp_{1^k,SCCMA}^{\text{SAV-}\Sigma}(A)$

$param \leftarrow \text{Setup}(1^k)$, $(pk, sk) \leftarrow \text{KeyGen}(1^k)$, $VString \leftarrow \text{SA-Ver-Setup}(param)$.

Then the challenger C sets $d \leftarrow 0$.

$(m^*, \sigma^*) \leftarrow A^{\text{O-SAV}}(pk, sk)$, where $\sigma^* \leftarrow \mathbb{Q} \setminus Q_{m^*}$ and Q_{m^*} denotes the set of valid signatures with respect to m^*.

Subsequently, the challenger plays the role of verifier to interact with A in **SA-Ver** protocol**:** $\langle \text{A}(param), \text{C}(m^*, \sigma^*, pk, VString) \rangle \rightarrow (\perp, b^*)$

If $b^* = 1$ then $d \leftarrow 1$

The advantage of A is $\Pr[Exp_{1^k,SCCMA}^{\text{SAV-}\Sigma}(A)] = 1$. The formalization of the oracle O-SAV is identical to that described in section 4.1.

Definition 2: An adversary A is said to (t, q_v, ε) strongly break soundness of SA-Ver in $\text{SAV-}\Sigma$ if A runs in time at most t, makes at most q_v calls to the oracle O-SAV, and succeeds with advantage ε in experiment $Exp_{1^k,SCCMA}^{\text{SAV-}\Sigma}(A)$.

SAV-Waters scheme has been proven to be secure against collusion and adaptive chosen message attack even for a computationally unbounded adversary. In the following, we show that even a computationally bounded adversary can break the strong soundness of SAV-Waters when allowed to choose the invalid signature for the challenge message by itself. So the work of Wu et al. [13] left an open question to design a server-aided verification protocol which meets the requirement of security against strong collusion attack for a signature scheme without random oracles. The weakness of SAV-Waters can be described as follows:

Given $sk = x$, the adversary A outputs a message m^* and chooses a random element $\sigma^* \leftarrow \mathbb{Q} \setminus Q_{m^*}$, where σ^* is an invalid signature of m^*. Then A(playing the role of server) runs the **SA-Ver** protocol to interact with C (on behalf of verifier) as follows:

(1) C takes $m^*, \sigma^* = (\sigma_1^*, \sigma_2^*), pk$ and $VString = e(g, g)$ as input, and computes $\overline{\sigma_1^*} = \sigma_1^* \cdot g^d$, where $d \leftarrow_R Z_p$.

$C \to A$: $(m, \overline{\sigma_1^*}, \sigma_2)$.

(2) At first A computes a valid signature for m^* by sk:

$$\lambda_1 = g^x (V_0 \prod_{i=1}^{n} V_i^{m_i^*})^s \ , \ \lambda_2 = g^s$$

Then A computes $\overline{K_2} = e(\overline{\sigma_1^*} \cdot \lambda_1, g)$, $\overline{K_3} = e(V_0 \prod_{i=1}^{n} V_i^{m_i^*}, \lambda_2) \cdot e(\overline{\sigma_1^*}, g)$

(3) C computes $pk_2 \bullet \overline{K_3} \bullet (K_1)^d = e(g,g)^x \bullet \ e(V_0 \prod_{i=1}^{n} V_i^{m_i^*}, \lambda_2) \bullet e(\sigma_1^*, g) \ \bullet e(g,g)^d$

$$= pk_2 \bullet \ e(V_0 \prod_{i=1}^{n} V_i^{m_i^*}, \lambda_2) \bullet e(\overline{\sigma_1^*}, g) = e(\overline{\sigma_1^*} \cdot \lambda_1, g) = \overline{K_2}$$

5. SAV-Hofheinz

We construct a SAV protocol called SAV-Hofheinz based on Hofheinz short signature [6] whose unforgeability was proven in the standard model by programmable hash function.

1.Setup: Given the security parameter k, $PG(1^k)$ outputs (e, G, G_T, g, p). Let n, η be two polynomials in k and H=(H.Gen, H.Eval, H.TrapGen, H.TrapEval) be a programmable hash function over group G with inputs from $\{0,1\}^n$. The common parameter is $param = (e, G, G_T, g, p, H)$.

2. KeyGen: Generates a hash function key through $K \leftarrow \text{H.Gen}(1^k)$. Pick $x \leftarrow_R Z_p^*$ and compute $X = g^x \in G$. The public key pk is $(pk_1 = K, pk_2 = X)$. The secret key sk is x.

3. Sig: Given a message $m \in \{0,1\}^n$, pick $s \leftarrow_R \{0,1\}^\eta$ and compute $y = H_K(m)^{\frac{1}{x+s}}$. The signature $\sigma = (s, y)$.

4. Ver: Given the public key pk, a message m and the signature $\sigma = (s, y)$, outputs 1 if s is of length η and $e(y, X \cdot g^s) = e(H_K(m), g)$.

5. SA-Ver-Setup($param$): A verifier takes **param** as input an computes $K_1 = e(g, g)$. $VString = K_1$.

6. SA-Ver: The verifier (resp., Server) takes $(m, (s, y), pk, VString)$ (resp., **param**) as input. Then the **SA-Ver** protocol is run as follows:

Offline Phase:

(1) The verifier picks $u, v \leftarrow_R Z_p$ and computes $V = g^u, U = (K_1)^v$.

Remark: As g and K_1 are fixed parameters for a verifier and will be exponentiated repeatedly, we can pick u, v and compute $V = g^u, U = (K_1)^v$ in advance since these values are independent of the received signature. So the computation time during online phase can be reduced by performing pre-computation during offline phase. The PBC library [10] provides a function called element_pp_pow to support fixed-base exponentiation algorithm [9].

Online Phase:

(1) The verifier computes $\theta = H_K(m)^u \cdot V$: Verifier → Server: θ

(2) The server computes $K_2 = e(\theta, g)$: Server → Verifier: K_2

(3) The verifier outputs 1 if $K_2 = e(y, X \cdot g^s)^u \cdot U$.

Theorem 1: Any adversary can only $(t, q_v, \frac{1}{p})$ break the strong soundness of SA-Ver in SAV-Hofheinz .

Proof: Having obtained (pk, sk), the adversary A picks an invalid signature (σ^*, s^*) for the challenge message m^*.

Then the challenger C takes $m^*, \sigma^* = (s^*, y^*)$ as input. Subsequently, A(playing the role of server) runs the **SA-Ver** protocol with C (on behalf of verifier) as follows:

The challenger C computes $\theta^* = H_K(m^*)^{u^*} \cdot g^{v^*}$, where $u^*, v^* \leftarrow_R Z_p$ and sends θ^* to the adversary. From the view of a computationally unbounded adversary, we have the following equation:

$$\log_{H_K(m^*)} \theta^* = u^* + v^* \cdot \log_{H_K(m^*)} g \bmod p \quad (1)$$

Assume the adversary returns K_2^* such that $K_2^* = e(y^*, X \cdot g^{s^*})^{u^*} \cdot (K_1)^{v^*}$. Then

$$\log_{e(H_K(m^*),g)} K_2^* = u^* \cdot \log_{e(H_K(m^*),g)} e(y^*, X \cdot g^{s^*}) + v^* \cdot \log_{e(H_K(m^*),g)} K_1 \bmod p \quad (2)$$

As $K_1 = e(g, g)$, we have $\log_{e(H_K(m^*),g)} K_1 = \log_{H_K(m^*)} g$ and

$$\begin{bmatrix} 1 & \log_{H_K(m^*)} g \\ \log_{e(H_K(m^*),g)} e(y^*, X \cdot g^{s^*}) & \log_{H_K(m^*)} g \end{bmatrix} \begin{bmatrix} u^* \\ v^* \end{bmatrix} = \begin{bmatrix} \log_{H_K(m^*)} \theta^* \\ \log_{e(H_K(m^*),g)} K_2^* \end{bmatrix} \quad (3)$$

As σ^* is an invalid signature, $\log_{e(H_K(m^*),g)} e(y^*, X \cdot g^{s^*}) \neq 1$ and the determinant of the above matrix is not equal to 0. Hence (u^*, v^*) can be uniquely determined from the view of the adversary.

However, by the equation $\theta^* = H_K(m^*)^{u^*} \cdot g^{v^*}$, the correct value of (u^*, v^*) is hidden from the adversary's view since there are p possible solutions. Consequently, the adversary returns K_2^* such that $K_2^* = e(y, X \cdot g^{s^*})^{u^*} \cdot (K_1)^{v^*}$ only with probability $\frac{1}{p}$.

6. Performance comparison

The number of operations for a verifier is listed in Table 1. We implement Hofheinz signature and SAV-Hofheinz with the PBC library [10], which is a free C library that performs rapid pairing operations. We take waters' group hash function $H_K(m) = V_0 \prod_{i=1}^{n} V_i^{m_i}$ as an instance of programmable hash function [6], where $m \in \{0,1\}^n, m = m_1 \cdots m_n$, $K = (V_0, V_1, \cdots, V_n) \in G$. The implementation results are recorded in Table 3. The result is obtained on a computer equipped with Intel Atom N270 Processor, 1.6 GHz and 1 GB memory, under the cygwin 1.7.9-1 platform. We use type-A symmetric pairing provided by the PBC library to implement the above schemes. Table 2 provides concise description of type-A pairing parameters provided by a.param and a1.param.

Table 1 The number of operations for a verifier

Scheme		Verification cost of the verifier		
		Pairing	Exponentiation	Multiplication
Hofheinz		2	1 (G)	3(G)
SAV-Hofheinz	Online	1	2 (G) +1 (G_T)	3(G)
	Offline	0	1(G) +1 (G_T)	0

Table 2 PBC Parameter description

PBC Parameter type	embedding degree	logp(in bits)	Representation size(in bits)	
			G	G_T
a.param	2	160	512	1024
a1.param	2	160	1024	2048

Table 3 Running time of verification

Scheme		Running time of verification	
		Implemented with a.param	Implemented with a1.param
Hofheinz		75.6ms	1867.8ms
SAV-Hofheinz	Offline	3.1ms	61.3ms
	Online	67.7ms	1375.5ms

7. Conclusion

In this paper, we design a server-aided verification signature SAV-Hofheinz based on a short signature proposed by Hofheinz [6]. The online computation in SAV-Hofheinz can be speeded up by pre-computing some exponentiation operations during the offline phase. In the following, we prove that our server-aided verification protocol meets the requirement of security against strong collusion attack defined in [13]. Finally we implement Hofheinz short signature and SAV-Hofheinz by the PBC library [10] and show that the running time of a verifier in SAV-Hofheinz is strictly less than that of a verifier in the verification protocol of Hofheinz signature.

Acknowledgement

This work is supported by Natural Science Foundation of Higher Education Institutions, in Jiangsu Province office of education, P.R. China (Grant No. 10KJD520005), National Natural Science Foundation of China(Grant No. 60803122), Innovative Foundation of Yangzhou University (Grant No. 2011CXJ022, 2011CXJ023).

References

[1] D. Boneh and X. Boyen, “Short Signatures without Random Oracles”, Eurocrypt ’04, volume 3027 of Lecture Notes in Computer Science, pages 382–400. Springer-Verlag, 2004

[2] D. Boneh, G. Lynn, H. Shacham, “Short Signature from The Weil Pairing”, Journal of Cryptology, 17(2004), pp. 297–319

[3] M. De Soete and J. J. Quisquater, “Speeding Up Smart Card RSA Computations with Insecure Coprocessors”, Smart Card 2000, pp. 191–197, 1989

[4] M. Girault, D. Lefranc, “Server-aided verification: theory and practice”, ASIACRYPT’05, Lecture Notes in Computer Science, vol. 3788, pp. 605–623, 2005

[5] M. Girault and J. J. Quisquater, “GQ + GPS = new ideas + new protocols”, Eurocrypt ’02 - Rump Session, 2002

[6] D. Hofheinz, E. Kiltz, ”Programmable Hash Functions and Their Applications”, Journal of Cryptology, to be published

[7] S. Hohenberger and A. Lysyanskaya, “How to Securely Outsource Cryptographic Computations”, TCC 2005, LNCS 3378, pp. 264-282, 2005

[8] C. H. Lim and P. J. Lee, “Server (prover/signer)-Aided Verification of Identity Proofs and Signatures”, Eurocrypt ’95, volume 921 of Lecture Notes in Computer Science, pp. 64–78, 1995

[9] A.J. Menezes, P.C.van Oorschot and S.A. Vanstone, “Handbook of Applied Cryptography”, CRC Press, 1997

[10] PBC library, http://crypto.stanford.edu/pbc/

[11] B. Waters, “Efficient identity-based encryption without random oracles”, in: In EUROCRYPT 2005, in: Lecture Notes in Computer Science, vol. 3494, pp. 114–127, 2005

[12] W. Wu, Y. Mu, W. Susilo, X. Huang, “Server-Aided Verification Signatures: Definitions and New Constructions”, in: ProvSec 2008, Lecture Notes in Computer Science, vol. 5324, Springer-Verlag, pp. 141–155, 2008

[13] Wei Wu, Yi Mu, Willy Susilo, Xinyi Huang, “Provably secure server-aided verification signatures”, Computers and Mathematics with Applications, Vol.61, pp.1705–1723, 2011

[14] F. Zhang, R. Safavi-Naini, and W. Susilo, “An Efficient Signature Scheme from Bilinear Pairing and its Applications”, Public Key Cryptography, volume 2947 of Lecture Notes in Computer Science, pp. 277–290, 2004.