1-1-2015

Efficient algorithms for secure outsourcing of bilinear pairings

Xiaofeng Chen
Xidian University

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

Jin Li
Guangzhou University

Duncan Wong
City University of Hong Kong, dwong@uow.edu.au

Jianfeng Ma
Xidian University

See next page for additional authors

Recommended Citation
Chen, Xiaofeng; Susilo, Willy; Li, Jin; Wong, Duncan; Ma, Jianfeng; Tang, Shaohua; and Tang, Qiang, "Efficient algorithms for secure outsourcing of bilinear pairings" (2015). *Faculty of Engineering and Information Sciences - Papers: Part A*. 3262.
https://ro.uow.edu.au/eispapers/3262

Efficient algorithms for secure outsourcing of bilinear pairings

Abstract

The computation of bilinear pairings has been considered the most expensive operation in pairing-based cryptographic protocols. In this paper, we first propose an efficient and secure outsourcing algorithm for bilinear pairings in the two untrusted program model. Compared with the state-of-the-art algorithm, a distinguishing property of our proposed algorithm is that the (resource-constrained) outsourcer is not required to perform any expensive operations, such as point multiplications or exponentiations. Furthermore, we utilize this algorithm as a subroutine to achieve outsource-secure identity-based encryptions and signatures.

Keywords

Cloud computing, Outsource-secure algorithms, Bilinear pairings, Untrusted program model

Disciplines

Engineering | Science and Technology Studies

Publication Details

Xiaofeng Chen, X., Susilo, W., Li, J., Wong, D., Ma, J., Tang, S. and Tang, Q. (2015). Efficient algorithms for secure outsourcing of bilinear pairings. Theoretical Computer Science, 562 (January), 112-121.

Authors

Xiaofeng Chen, Willy Susilo, Jin Li, Duncan Wong, Jianfeng Ma, Shaohua Tang, and Qiang Tang

Efficient Algorithms for Secure Outsourcing of Bilinear Pairings

Xiaofeng Chen [a],Willy Susilo [b],Jin Li [c],Duncan S. Wong [d],

Jianfeng Ma [a],Shaohua Tang [e],Qiang Tang [f]

[a] *State Key Laboratory of Integrated Service Networks (ISN),*

Xidian University, Xi'an, P.R.China

[b] *Centre for Computer and Information Security Research (CCISR)*

School of Computer Science and Software Engineering

University of Wollongong, Australia

[c] *School of Computer Science and Educational Software*

Guangzhou University, Guangzhou, P.R. China

[d] *Department of Computer Science,*

City University of Hong Kong, Hong Kong

[e] *School of Computer Science and Engineering*

South China University of Technology, Guangzhou, P.R. China

[f] *APSIA group, SnT, University of Luxembourg*

6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg

Abstract

The computation of bilinear pairings has been considered the most expensive operation in pairing-based cryptographic protocols. In this paper, we first propose an efficient and secure outsourcing algorithm for bilinear pairings in the two untrusted program model. Compared with the state-of-the-art algorithm, a distinguishing property of our proposed algorithm is that the (resource-constrained) outsourcer is not required to perform any expensive operations, such as point multiplications or exponentiations. Furthermore, we utilize this algorithm as a subroutine to achieve outsource-secure identity-based encryptions and signatures.

Key words: Cloud computing, Outsource-secure algorithms, Bilinear pairings, Untrusted program model.

1 Introduction

With the rapid development in availability of cloud services, the techniques for securely outsourcing the prohibitively expensive computations to untrusted servers are getting more and more attentions in the scientific community. In the outsourcing computation paradigm, the resource-constrained devices can enjoy the unlimited computation resources in a pay-per-use manner, which avoids large capital outlays in hardware/software deployment and maintenance.

Despite the tremendous benefits, outsourcing computation also inevitably introduces some new security concerns and challenges. Firstly, the computation tasks often contain some sensitive information that should not be exposed to the untrusted cloud servers. Therefore, the first security challenge is the *secrecy* of the outsourcing computation: the cloud servers should not learn anything about the data (including the *secret* inputs and the outputs). We argue that the encryption can only provide a partial solution to this problem since it is very difficult to perform meaningful computations over the encrypted data. Note that fully homomorphic encryption could be a potential solution, but the

* The corresponding author: Xiaofeng Chen (xfchen@xidian.edu.cn)

existing schemes are impractical. Secondly, the semi-trusted cloud servers may return an invalid result. For example, the servers might contain a software bug that will fail on a constant number of invocations. Moreover, the servers might decrease the amount of the computation due to financial incentives and then return a computationally indistinguishable (invalid) result. Therefore, the second security challenge is the *checkability* of the outsourcing computation: the outsourcer should have the ability to detect any failures if the cloud servers misbehave. Trivially, the test procedure should never need to perform other complicated computations since the computationally limited devices such as RFID tags or smartcard may be incapable to accomplish the test. At the very least, it must be *far more* efficient than accomplishing the computation task itself (recall the motivation for outsourcing computations).

In the last decade, the bilinear pairings, especially the Weil pairing and Tate pairing of algebraic curves, have initiated some completely new fields in cryptography, making it possible to realize cryptographic primitives that were previously unknown or impractical [11,15,34]. Trivially, implementing the pairing-based cryptographic protocols is dependent on the fast computation of pairings, and thus plenty of research work has been done to implement this workload efficiently [10,13,15,33,36,42].

The computation of bilinear pairings has been considered the prohibitive expensive operation in embedded devices such as the RFID tag or smardcard (note that we even assume that the modular exponentiation is too expensive to be carried out on such devices). Chevallier-Mames et al. [20] presented the first algorithm for secure delegation of elliptic-curve pairings based on an untrusted server model. Besides, the outsourcer could detect any failures with probability 1 if the server misbehaves. However, an obvious disadvantage of the algorithm is that the outsourcer should carry out some other expensive operations such as point multiplications and exponentiations. More precisely, on the one hand, we argue that these expensive operations might be too resource consuming to be carried out on a computationally limited device. On the other hand, the computation of point multiplications is even comparable to that of bilinear pairings in some scenarios [25,42] [1]. Therefore, it is meaningless if the client must perform point multiplications in order to outsource pairings since this contradicts with the aim of outsourcing computation. Therefore, the algorithm is meaningless for real-world applications in this sense. To the best of our knowledge, it seems that all of the following works on delegation of bilinear pairings [17,35,44] also suffer from the same problems.

Our Contribution. In this paper, we propose the first efficient and secure outsourcing algorithm of bilinear pairings in the one-malicious version of two untrusted program model [32]. Compared with the state-of-the-art algorithm in [20], a distinguishing property of our proposed algorithm is that the (resource-constrained) outsourcer never needs to perform any expensive operations such as point multiplications and exponentiations. Hence, our proposed algorithm is very practical. Furthermore, we also utilize this algorithm as a subroutine to achieve outsource-secure Boneh-Franklin identity-based encryptions and Cha-Cheon identity-based signatures.

[1] As pointed out in [25,42], when the supersingular elliptic curve is defined over a 512-bit finite field with embedding degree 2, the computational overhead of a point multiplication is almost the same as that of a standard Tate pairing.

1.1 Related Work

Abadi et al. [2] proved the impossibility of secure outsourcing an exponential computation while locally doing only polynomial time work. Therefore, it is meaningful only to consider outsourcing expensive polynomial time computations. The theoretical computer science community has devoted considerable attention to the problem of how to securely outsource different kinds of expensive computations. Atallah et al. [3] presented a framework for secure outsourcing of scientific computations such as matrix multiplications and quadrature. However, the solution used the disguise technique and thus allowed leakage of private information. Atallah and Li [4] investigated the problem of computing the edit distance between two sequences and presented an efficient protocol to securely outsource sequence comparisons to two servers. Recently, Blanton et al. proposed a more efficient scheme for secure outsourcing sequence comparisons [9]. Blanton and Aliasgari [6,7] proposed an efficient scheme for secure outsourcing DNA computations and biometric comparisons. Benjamin and Atallah [5] addressed the problem of secure outsourcing for widely applicable linear algebra computations. However, the proposed protocols required the expensive operations of homomorphic encryptions. Atallah and Frikken [1] further studied this problem and gave improved protocols based on the so-called weak secret hiding assumption. Recently, Wang et al. [45] presented efficient mechanisms for secure outsourcing of linear programming computations.

The problem of securely outsourcing expensive computations has been well studied in the cryptography community. In 1992, Chaum and Pedersen [21] firstly introduced the notion of wallets with observers, a piece of secure hardware installed on the client's computer to perform some expensive computations. Hohenberger and Lysyanskaya [32] proposed the first outsource-secure algorithm for modular exponentiations based on the two previous approaches of precomputation [16,41] and server-aided computation [29,39]. Very recently, Chen et al. [19] proposed more efficient outsource-secure algorithms for (simultaneously) modular exponentiation in the two untrusted program model.

Since the servers (or workers) are not trusted by the outsourcers, Golle and Mironov [31] first introduced the concept of ringers to solve the trust problem of verifying computation completion. The following works focused on the other trust problem of retrieving payments [8,23,24,43]. Besides, Gennaro et al. [27] first formalized the notion of verifiable computation to solve the problem of verifiably outsourcing the computation of an arbitrary functions, which has attracted the attention of plenty of researchers [14,28,30,37,38]. Gennaro et al. [27] also proposed a protocol that allowed the outsourcer to efficiently verify the outputs of the computations with a computationally sound, *non-interactive* proof (instead of interactive ones). Benabbas et al. [12] presented the first practical verifiable computation scheme for high degree polynomial functions. In 2011, Green et al. [26] proposed new methods for efficiently and securely outsourcing decryption of attribute-based encryption (ABE) ciphertexts. Based on this work, Parno et al. [40] showed a construction of a multi-function verifiable computation scheme.

1.2 Organization

The rest of the paper is organized as follows. Some background and preliminaries that will be required throughout this paper are presented in Section

2. The security definitions for outsourcing computation are provided in Section 3. The proposed new outsource-secure bilinear pairings algorithm and its security analysis are presented in Section 4. The proposed outsource-secure identity-based encryptions and signatures are given in Section 5. Finally, Section 6 concludes the paper.

2 Preliminaries

In this section, we will briefly describe the basic definition and properties of bilinear pairings [11,15,18,25] and then overview the algorithm for delegation of pairings [20].

2.1 Bilinear Pairings

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic additive groups generated by $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. The order of $\mathbb{G}_1$ and $\mathbb{G}_2$ is a large prime order q. Define $\mathbb{G}_T$ to be a cyclic multiplicative group of the same order q. A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with the following properties:

(1) Bilinear: $e(aR, bQ) = e(R, Q)^{ab}$ for all $R \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_q^*$.
(2) Non-degenerate: There exists $R \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$ such that $e(R, Q) \neq 1$.
(3) Computable: There is an efficient algorithm to compute $e(R, Q)$ for all $R, Q \in \mathbb{G}_1$.

The examples of such groups can be found in supersingular elliptic curves or hyperelliptic curves over finite fields, and the bilinear pairings can be derived from the Weil or Tate pairings. For more details, see [11,15,18,25].

For the ease of simplicity, we use the above notations throughout this paper.

2.2 Algorithm for Delegation of Elliptic-Curve Pairings

The input of Chevallier-Mames et al.'s algorithm [20] is two random points $A \in \mathbb{G}_1$, $B \in \mathbb{G}_2$, and the output is $e(A, B)$. Assume that the outsourcer T has been given the value of $e(\mathcal{P}_1, \mathcal{P}_2)$.

(1) The outsourcer T generates two random elements $g_1, g_2 \in \mathbb{Z}_q$, and queries the following pairings to the server U:

$$\alpha_1 = e(A + g_1\mathcal{P}_1, \mathcal{P}_2), \alpha_2 = e(\mathcal{P}_1, B + g_2\mathcal{P}_2), \alpha_3 = e(A + g_1\mathcal{P}_1, B + g_2\mathcal{P}_2).$$

(2) The outsourcer T verifies that $\alpha_i \in \mathbb{G}_T$, by checking $\alpha_i^q = 1$ for $i = 1, 2, 3$. Otherwise, T outputs $\perp$ and halts.

(3) The outsourcer T computes $e(A, B) = \alpha_1^{-g_2}\alpha_2^{-g_1}\alpha_3 e(\mathcal{P}_1, \mathcal{P}_2)^{g_1 g_2}$.

(4) The outsourcer T generates four random elements $a_1, r_1, a_2, r_2 \in \mathbb{Z}_q$, and queries the following pairing to the server U:

$$\alpha_4 = e(a_1 A + r_1\mathcal{P}_1, a_2 B + r_2\mathcal{P}_2).$$

(5) The outsourcer T computes

$$\alpha_4' = e(A, B)^{a_1 a_2}\alpha_1^{a_1 r_2}\alpha_2^{a_2 r_1} e(\mathcal{P}_1, \mathcal{P}_2)^{r_1 r_2 - a_1 g_1 r_2 - a_2 g_2 r_1}.$$

T outputs $e(A, B)$ if and only if $\alpha_4' = \alpha_4$.

Remark 1. We argue that the outsourcer T should perform some expensive operations such as point multiplications and exponentiations. In some cases, this contradicts with the motivation of the outsourcing computations.

3 Formal Security Definitions

In this section, we introduce some definitions for secure outsourcing of a cryptographic algorithm [32].

Informally, we say that an honest but resources-constrained component T securely outsources some expensive work to an untrusted component U, and (T, U) is an *outsource-secure* implementation of a cryptographic algorithm Alg if (1) T and U implement Alg, i.e., $\mathsf{Alg} = T^U$ and (2) suppose that T is given oracle access to a malicious U' (instead of U) that records all of its computation over time and tries to act maliciously, U' cannot learn anything interesting about the input and output of $T^{U'}$. Besides, another part of the adversary $\mathcal{A}$ is the adversarial environment E that submits adversatively chosen inputs to Alg, i.e., $\mathcal{A} = (E, U')$. One fundamental assumption is that E and U' will not have a direct communication channel after they begin interacting with T (although E and U' may develop a joint strategy beforehand). That is, E and U' can only communicate with each other by passing messages through T. In the real world, a malicious manufacturer E might program its software U' to behave in an adversarial fashion. However, once U' has been installed behind the firewall of T, E is no longer able to send instructions to U'. This implies that E may know something about the protected inputs to Alg that U' does not. For example, E can see all of its own adversarial inputs to Alg, while T might hide some of these from U'. Otherwise, if U' could see any values chosen by E, then E and U' still can agree on a joint strategy that causes U' to terminate its tasks upon receiving some predefined message from E. As a result, no security guarantee can be provided. We illustrate this with the proposed outsourcing algorithm [19], if E could capture all of network traffic of T, then E can know which are the test queries (note that T must invoke the subroutine *Rand* and store all the results in its hard disk). As a result, U' can also know the facts by communicating with E. Consequently, when T sends the queries to U', U' only honestly computes the results for the test queries. For the remaining queries, U' terminates and just returns a random value. Therefore, U' can always cheat T without being detected and no security guarantees can be obtained.

The inputs to Alg can be categorized into three logical divisions: (1) Secret: information is only available to T (e.g., a secret key or a plaintext) and remains hidden from E and U'; (2) Protected: information is only available to T and E (e.g., a public key or a ciphertext) while remains hidden from U'; (3) Unprotected: information is available to T, E and U' (e.g, the time-stamp). similarly, Alg has secret, protected, and unprotected outputs. Moreover, the divisions for inputs can be further categorized based on whether the inputs are generated honestly or adversarially except the case of adversarial, secret inputs (note that E cannot generate secret inputs which are only available to T). Therefore, Alg will take five types of inputs and produce three types of outputs.

The formal definition of an algorithm with outsource-input/output is given as follows:

Definition 1 ***(Algorithm with outsource-I/O)*** *An algorithm* Alg *obeys the outsource input/output specification if it takes five inputs, and produces three outputs. The first three inputs are generated by an honest party, and are classified by how much the adversary* $\mathcal{A} = (E, U')$ *knows about them, where* E *is the adversarial environment that submits adversarially chosen inputs to*

Alg, and U' is the adversarial software operating in place of oracle U. The first input is called the honest, secret input, which is unknown to both E and U'; the second is called the honest, protected input, which may be known by E, but is protected from U'; and the third is called the honest, unprotected input, which may be known by both E and U. In addition, there are two adversarially-chosen inputs generated by the environment E: the adversarial, protected input, which is known to E, but protected from U'; and the adversarial, unprotected input, which may be known by E and U[2]*. Similarly, the first output called secret is unknown to both E and U'; the second is protected, which may be known to E, but not U'; and the third is unprotected, which may be known by both parties of $\mathcal{A}$.*

The following definition of outsource-security means that if a malicious U' can learn something secret or protected about the inputs to T^U from being T's oracle instead of U, it can also learn without that. That is, there exists a simulator S that, when told that $T^U(x)$ was invoked, simulates the view of U' without access to the secret or protected inputs of x. Similarly, the definition also ensures that the malicious environment E cannot gain any knowledge of the secret inputs and outputs of T^U, even if T uses the malicious software U' written by E. Also, there exists a simulator S' that, when told that $T^U(x)$ was invoked, can simulate the view of E without access to the secret inputs of x.

Definition 2 ***(Outsource-security)*** *Let Alg be an algorithm with outsource I/O. A pair of algorithms (T, U) is said to be an outsource-secure implemen-*

[2] For any outsource-secure implementation in the real applications, the adversarial, unprotected input must be empty. Even if it contains a single bit, then a covert channel may be created from E and U'. Then, a k bits of shared information can be obtained after interacting k rounds.

tation of Alg *if:*

(1) Correctness: T^U *is a correct implementation of* Alg.

(2) Security: For all probabilistic polynomial-time adversaries $\mathcal{A} = (E, U')$, *there exist probabilistic expected polynomial-time simulators* (S_1, S_2) *such that the following pairs of random variables are computationally indistinguishable.*

- *Pair One.* $\mathrm{EVIEW}_{\mathrm{real}} \sim \mathrm{EVIEW}_{\mathrm{ideal}}$*:*
  - *The view that the the adversarial environment* E *obtains by participating in the following* real *process:*

$$\mathrm{EVIEW}^i_{\mathrm{real}} = \{(\mathrm{istate}^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^k, \mathrm{istate}^{i-1});$$

$$(\mathrm{estate}^i, j^i, x^i_{ap}, x^i_{au}, \mathrm{stop}^i) \leftarrow E(1^k, \mathrm{EVIEW}^{i-1}_{\mathrm{real}}, x^i_{hp}, x^i_{hu});$$

$$(\mathrm{tstate}^i, \mathrm{ustate}^i, y^i_s, y^i_p, y^i_u) \leftarrow$$

$$T^{U'(\mathrm{ustate}^{i-1})}(\mathrm{tstate}^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}) :$$

$$(\mathrm{estate}^i, y^i_p, y^i_u)\}$$

$$\mathrm{EVIEW}_{\mathrm{real}} = \mathrm{EVIEW}^i_{\mathrm{real}} \text{ if } \mathrm{stop}^i = \mathrm{TRUE}.$$

The real process proceeds in rounds. In round i*, the honest (secret, protected, and unprotected) inputs* $(x^i_{hs}, x^i_{hp}, x^i_{hu})$ *are picked using an honest, stateful process* I *to which the environment* E *does not have access. Then* E*, based on its view from the last round, chooses (0) the value of its* estate_i *variable as a way of remembering what it did next time it is invoked; (1) which previously generated honest inputs* $(x^i_{hs}, x^i_{hp}, x^i_{hu})$ *to give to* $T^{U'}$ *(note that* E *can specify the index* j^i *of these inputs, but not their values); (2) the adversarial, protected input* x^i_{ap}*; (3) the adversarial, unprotected input* x^i_{au}*; (4) the Boolean variable* stop^i *that determines whether round* i *is the last round in this process. Next, the algorithm* $T^{U'}$ *is run on the inputs* $(\mathrm{tstate}^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$, *where* tstate^{i-1} *is* T*'s previously saved state, and produces a new state* tstate^{i} *for* T*, as well as the secret* y_s^i*, protected* y_p^i *and unprotected* y_u^i *outputs. The oracle* U' *is given its previously saved state,* ustate^{i-1}*, as input, and the current state of* U' *is saved in the variable* ustate^{i}*. The view of the real process in round* i *consists of* estate^{i}*, and the values* y_p^i *and* y_u^i*. The overall view of* E *in the real process is just its view in the last round (i.e.,* i *for which* $\mathrm{stop}^i = \mathrm{TRUE}$*.).*

- *The* ideal *process:*

$$
\begin{aligned}
\mathrm{EVIEW}^i_{\mathrm{ideal}} = \{ & (\mathrm{istate}^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^k, \mathrm{istate}^{i-1}); \\
& (\mathrm{estate}^i, j^i, x^i_{ap}, x^i_{au}, \mathrm{stop}^i) \leftarrow E(1^k, \mathrm{EVIEW}^{i-1}_{\mathrm{ideal}}, x^i_{hp}, x^i_{hu}); \\
& (\mathrm{astate}^i, y^i_s, y^i_p, y^i_u) \leftarrow \mathsf{Alg}(\mathrm{astate}^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}); \\
& (\mathrm{sstate}^i, \mathrm{ustate}^i, Y^i_p, Y^i_u, \mathrm{rep}^i) \leftarrow S_1^{U'(\mathrm{ustate}^{i-1})} \\
& (\mathrm{sstate}^{i-1}, \cdots, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}, y^i_p, y^i_u); \\
& (z^i_p, z^i_u) = \mathrm{rep}^i(Y^i_p, Y^i_u) + (1 - \mathrm{rep}^i)(y^i_p, y^i_u) : \\
& (\mathrm{estate}^i, z^i_p, z^i_u) \}
\end{aligned}
$$

$\mathrm{EVIEW}_{\mathrm{ideal}} = \mathrm{EVIEW}^i_{\mathrm{ideal}}$ *if* $\mathrm{stop}^i = \mathrm{TRUE}$.

The ideal process also proceeds in rounds. In the ideal process, we have a stateful simulator S_1 *who, shielded from the secret input* x^i_{hs}*, but given the non-secret outputs that* Alg *produces when run all the inputs for round* i*, decides to either output the values* (y^i_p, y^i_u) *generated by* Alg*, or replace them with some other values* (Y^i_p, Y^i_u)*. Note that this is captured by having the indicator variable* rep^i *be a bit that determines whether* y^i_p *will be replaced with* Y^i_p*. In doing so, it is allowed to query oracle* U'*; moreover,* U' *saves its state as in the real experiment.*

- *Pair Two.* $\mathrm{UVIEW}_{\mathrm{real}} \sim \mathrm{UVIEW}_{\mathrm{ideal}}$*:*
  - *The view that the untrusted software* U' *obtains by participating in*

the real *process described in Pair One.* $\mathrm{UVIEW}_{\mathrm{real}} = (\mathrm{ustate}^i, y_u^i)$ *if* $\mathrm{stop}^i = \mathrm{TRUE}$.

· *The* ideal *process:*

$$\mathrm{UVIEW}^i_{\mathrm{ideal}} = \{(\mathrm{istate}^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^k, \mathrm{istate}^{i-1});$$

$$(\mathrm{estate}^i, j^i, x^i_{ap}, x^i_{au}, \mathrm{stop}^i) \leftarrow E(1^k, \mathrm{estate}^{i-1}, x^i_{hp}, x^i_{hu}, y^{i-1}_p, y^{i-1}_u);$$

$$(\mathrm{astate}^i, y^i_s, y^i_p, y^i_u) \leftarrow \mathsf{Alg}(\mathrm{astate}^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au});$$

$$(\mathrm{sstate}^i, \mathrm{ustate}^i) \leftarrow S_2^{U'(\mathrm{ustate}^{i-1})}\ (\mathrm{sstate}^{i-1}, x^{j^i}_{hu}, x^i_{au}, y^i_u) :$$

$$(\mathrm{ustate}^i, y^i_u)\}$$

$\mathrm{UVIEW}_{\mathrm{ideal}} = \mathrm{UVIEW}^i_{\mathrm{ideal}}$ *if* $\mathrm{stop}^i = \mathrm{TRUE}$.

In the ideal process, we have a stateful simulator S_2 who, equipped with only the unprotected inputs/outputs $(x^i_{hu}, x^i_{au}, y^i_u)$, queries U'. As before, U' may maintain state.

Given an outsource-secure implementation of a cryptographic algorithm $\mathsf{Alg} = T^U$, we should compare the overhead of T with that for the fastest known implementation of Alg. Besides, if the algorithm Alg could not provide 100 percent checkability, we should evaluate the probability that T could detect the misbehavior of U.

Definition 3 *(**α-efficient, secure outsourcing**) A pair of algorithms (T, U) is said to be an α-efficient implementation of Alg if (1) T^U is a correct implementation of Alg and (2) $\forall$ inputs x, the running time of T is no more than an α-multiplicative factor of the running time of Alg.*

Definition 4 *(**β-checkable, secure outsourcing**) A pair of algorithms (T, U) is said to be an β-checkable implementation of Alg if (1) T^U is a correct implementation of Alg and (2) $\forall$ inputs x, if U' deviates from its advertised functionality during the execution of $T^{U'}(x)$, T will detect the error with prob-*

ability no less than β.

Definition 5 *((α,β)-**outsource-security**) A pair of algorithms (T, U) is said to be an (α,β)-outsource-secure implementation of* Alg *if it is both α-efficient and β-checkable.*

4 New Outsource-Secure Algorithm of Bilinear Pairings

4.1 Security Model

Hohenberger and Lysyanskaya [32] first presented the so-called *two untrusted program model* for outsourcing cryptographic computations. In the two untrusted program model, the adversarial environment E writes the code for two (potentially different) programs $U' = (U'_1, U'_2)$. E then gives this software to T, advertising a functionality that U'_1 and U'_2 may or may not accurately compute, and T installs this software in a manner such that all subsequent communication between any two of E, U'_1 and U'_2 must pass through T. The new adversary attacking T is $\mathcal{A} = (E, U'_1, U'_2)$. Moreover, we assume that at most one of the programs U'_1 and U'_2 deviates from its advertised functionality on a non-negligible fraction of the inputs, while we cannot know which one and security means that there is a simulator $\mathcal{S}$ for both. This is named as the one-malicious version of two untrusted program model (i.e., “one-malicious model” for the simplicity) [3] . In the real-world applications, it is equivalent to

[3] Canetti, Riva, and Rothblum [22] introduced the refereed delegation of computation model, where the outsourcer delegates the computation to several servers under the assumption that at least one of the servers is honest. Trivially, one-malicious model can be viewed as a special case of refereed delegation of computation model.

buy the two copies of the advertised software from two different vendors and achieve the security as long as one of them is honest.

Similar to [32], we also use a subroutine named *Rand* in order to speed up the computations. The inputs for *Rand* are the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ with prime order q, the bilinear pairing e, and possibly some other (random) values, and the outputs for each invocation are a random, independent six-tuple $(V_1, V_2, v_1V_1, v_2V_1, v_2V_2,\ e(v_1V_1, v_2V_2))$, where $v_1, v_2 \in_R \mathbb{Z}_q^*$, $V_1 \in_R \mathbb{G}_1$, and $V_2 \in_R \mathbb{G}_2$. A naive approach to implement this functionality is for a trusted server to compute a table of random, independent six-tuple in advance and then load it into the memory of T. For each invocation of *Rand*, T just retrieves a new six-tuple in the table (the table-lookup method).

4.2 Outsourcing Algorithm

In this section, we propose a new secure outsourcing algorithm **Pair** for bilinear pairings in the one-malicious model. In **Pair**, T outsources its pairing computations to U_1 and U_2 by invoking the subroutine *Rand*. A requirement for **Pair** is that the adversary $\mathcal{A}$ cannot know any useful information about the inputs and outputs of **Pair**.

The input of **Pair** is two random points $A \in \mathbb{G}_1$, $B \in \mathbb{G}_2$, and the output of **Pair** is $e(A, B)$. Note that A and B may be secret or (honest/adversarial) protected and $e(A, B)$ is always secret or protected. Moreover, both A and B are computationally blinded to U_1 and U_2. We let $U_i(\Lambda_1, \Lambda_2) \rightarrow e(\Lambda_1, \Lambda_2)$ denote that U_i takes as inputs (Λ_1, Λ_2) and outputs $e(\Lambda_1, \Lambda_2)$, where $i = 1, 2$. The proposed outsourcing algorithm **Pair** consists of the following steps:

(1) To implement this functionality using U_1 and U_2, T firstly runs *Rand* to create a blinding six-tuple $(V_1, V_2, v_1V_1, v_2V_1, v_2V_2, e(v_1V_1, v_2V_2))$. We denote $\lambda = e(v_1V_1, v_2V_2)$.

(2) The main trick of **Pair** is to logically split A and B into random looking pieces that can be computed by U_1 and U_2. Without loss of generality, let $\alpha_1 = e(A+v_1V_1, B+v_2V_2)$, $\alpha_2 = e(A+V_1, v_2V_2)$, and $\alpha_3 = e(v_1V_1, B+V_2)$. Note that

$$\alpha_1 = e(A, B)e(A, v_2V_2)e(v_1V_1, B)e(v_1V_1, v_2V_2),$$

$$\alpha_2 = e(A, v_2V_2)e(V_1, v_2V_2),$$

$$\alpha_3 = e(v_1V_1, B)e(v_1V_1, V_2),$$

Therefore, $e(A, B) = \alpha_1\alpha_2^{-1}\alpha_3^{-1}\lambda^{-1}e(V_1, V_2)^{v_1+v_2}$.

(3) T then runs *Rand* to obtain two new six-tuple

$$(X_1, X_2, x_1X_1, x_2X_1, x_2X_2, e(x_1X_1, x_2X_2))$$

and

$$(Y_1, Y_2, y_1Y_1, y_2Y_1, y_2Y_2, e(y_1Y_1, y_2Y_2)).$$

(4) T queries U_1 in random order as

$U_1(A + v_1V_1, B + v_2V_2) \to e(A + v_1V_1, B + v_2V_2) = \alpha_1$;

$U_1(v_1V_1 + v_2V_1, V_2) \to e(V_1, V_2)^{v_1+v_2}$;

$U_1(x_1X_1, x_2X_2) \to e(x_1X_1, x_2X_2)$;

$U_1(y_1Y_1, y_2Y_2) \to e(y_1Y_1, y_2Y_2)$;

Similarly, T queries U_2 in random order as

$U_2(A + V_1, v_2V_2) \to e(A + V_1, v_2V_2) = \alpha_2$;

$U_2(v_1V_1, B + V_2) \to e(v_1V_1, B + V_2) = \alpha_3$;

$U_2(x_1X_1, x_2X_2) \to e(x_1X_1, x_2X_2)$;

$U_2(y_1Y_1, y_2Y_2) \rightarrow e(y_1Y_1, y_2Y_2)$;

(5) Finally, T checks that both U_1 and U_2 produce the correct outputs, i.e., $e(x_1X_1, x_2X_2)$ and $e(y_1Y_1, y_2Y_2)$ for the test queries. If not, T outputs "error"; otherwise, T can compute $e(A, B) = \alpha_1\alpha_2^{-1}\alpha_3^{-1}\lambda^{-1}e(V_1, V_2)^{v_1+v_2}$.

Remark 2. Given a random point P in $\mathbb{G}_1$ (or $\mathbb{G}_2$), T can compute the inverse point $-P$ easily. Therefore, T can query $U_2(A + V_1, -v_2V_2) \rightarrow e(A + V_1, -v_2V_2) = \alpha_2^{-1}$ and $U_2(-v_1V_1, B+V_2) \rightarrow e(-v_1V_1, B+V_2) = \alpha_3^{-1}$. Similarly, we can define the outputs of *Rand* be

$$(V_1, V_2, v_1V_1, v_2V_1, v_2V_2, e(v_1V_1, v_2V_2)^{-1}).$$

Therefore, T needs not to perform the inverse computation in $\mathbb{G}_T$.

4.3 Security Analysis

Theorem 1 *In the one-malicious model, the algorithms* $(T, (U_1, U_2))$ *are an outsource-secure implementation of* ***Pair****, where the input* (A, B) *may be honest, secret; or honest, protected; or adversarial, protected.*

Proof. The proof is similar to [32]. The correctness is trivial and we only focus on security. Let $\mathcal{A} = (E, U_1', U_2')$ be a PPT adversary that interacts with a PPT algorithm T in the one-malicious model.

Firstly, we prove Pair One $EVIEW_{real} \sim EVIEW_{ideal}$:

Note that we only consider three types of input (A, B): honest, secret; or honest, protected; or adversarial, protected. If the input (A, B) is anything other than honest, secret (this means that the input (A, B) is either honest, protected or adversarial, protected. Obviously, neither types of input (A, B)

is secret), then the simulation is trivial. That is, the simulator S_1 behaves the same way as in the real execution. Trivially, S_1 never requires to access the secret input since neither types of input (A, B) is secret.

If (A, B) is an honest, secret input, then the simulator S_1 behaves as follows: On receiving the input on round i, S_1 ignores it and instead makes four random queries of the form (P_j, Q_j) to both U'_1 and U'_2. S_1 randomly tests two outputs (i.e., $e(P_j, Q_j)$) from each program. If an error is detected, S_1 saves all states and outputs Y_p^i="error", $Y_u^i=\varnothing$, $rep^i=1$ (i.e., the output for ideal process is $(estate^i$, "error"$, \varnothing)$). If no error is detected, S_1 checks the remaining two outputs. If all checks pass, S_1 outputs $Y_p^i=\varnothing$, $Y_u^i=\varnothing$, $rep^i=0$ (i.e., the output for ideal process is $(estate^i, y_p^i, y_u^i)$); otherwise, S_1 selects a random element r and outputs $Y_p^i=r$, $Y_u^i=\varnothing$, $rep^i=1$ (i.e., the output for ideal process is $(estate^i, r, \varnothing)$). In either case, S_1 saves the appropriate states.

The input distributions to (U'_1, U'_2) in the real and ideal experiments are computationally indistinguishable. In the ideal experiment, the inputs are chosen uniformly at random. In the real experiment, each part of all queries that T makes to any one program in the step (4) of **Pair** is independently re-randomized and the re-randomization factors are also truly randomly generated by using naive table-lookup method [4]. We consider the following three possible cases:

Firstly, if (U'_1, U'_2) behave honest in the round i, then $EVIEW^i_{real} \sim EVIEW^i_{ideal}$ (this is because $T^{(U'_1, U'_2)}$ perfectly executes **Pair** in the real experiment and

[4] We argue that if v_1, v_2, V_1, and V_2 are random elements in $\mathbb{Z}_q^*$, $\mathbb{Z}_q^*$, $\mathbb{G}_1$, and $\mathbb{G}_2$, respectively, then the output of *Rand* is also a random, independent six-tuple $(V_1, V_2, v_1V_1, v_2V_1, v_2V_2, e(v_1V_1, v_2V_2))$.

S_1 simulates with the same outputs in the ideal experiment, i.e., rep^i=0). Secondly, if one of (U_1', U_2') is dishonest in the round i and it has been detected by both T and S_1 (with probability $\frac{1}{2}$), then it will result in an output of "error". Finally, we consider the case that the output of **Pair** is corrupted, i.e., one of (U_1', U_2') is dishonest in the round i while it is undetected (with probability $\frac{1}{2}$) by T. In the real experiment, the four outputs generated by (U_1', U_2') are multiplied together along with a random value λ^{-1} (see the step (5) of our algorithm **Pair**). Thus, the output of **Pair** looks random to the environment E. In the ideal experiment, S_1 also simulates with a random value $r \in \mathbb{G}_T$ as the output. Thus, $EVIEW^i_{real} \sim EVIEW^i_{ideal}$ even when one of (U_1', U_2') is dishonest. By the hybrid argument, we conclude that $EVIEW_{real} \sim EVIEW_{ideal}$.

Secondly, we prove Pair Two $UVIEW_{real} \sim UVIEW_{ideal}$:

The simulator S_2 always behaves as follows: On receiving the input on round i, S_2 ignores it and instead makes four random queries of the form (P_j, Q_j) to both U_1' and U_2'. Then S_2 saves its states and the states of (U_1', U_2'). E can easily distinguish between these real and ideal experiments (note that the output in the ideal experiment is never corrupted). However, E cannot communicate this information to (U_1', U_2'). This is because in the round i of the real experiment, T always re-randomizes its inputs to (U_1', U_2'). In the ideal experiment, S_2 always generates random, independent queries for (U_1', U_2'). Thus, for each round i we have $UVIEW^i_{real} \sim UVIEW^i_{ideal}$. By the hybrid argument, we conclude that $UVIEW_{real} \sim UVIEW_{ideal}$. ∎

Theorem 2 *In the one-malicious model, the algorithms $(T, (U_1, U_2))$ are an $(O(\frac{1}{n}), \frac{1}{2})$-outsource-secure implementation of **Pair**, where n is the bit length* *of the order q of bilinear groups.*

Proof. The proposed algorithm **Pair** makes 3 calls to *Rand* plus 5 point addition in $\mathbb{G}_1$ (or $\mathbb{G}_2$), and 4 multiplication in $\mathbb{G}_T$ in order to compute $e(A, B)$. Also, the computation for *Rand* is negligible when using the table-lookup method. On the other hand, it takes roughly $O(n)$ multiplications in resulting finite filed to compute the bilinear pairing [5] . Thus, the algorithms $(T, (U_1, U_2))$ are an $O(\frac{1}{n})$-efficient implementation of **Pair**.

On the other hand, U_1 (resp. U_2) cannot distinguish the two test queries from the two real queries that T makes. If U_1 (resp. U_2) fails during any execution of **Pair**, it will be detected with probability $\frac{1}{2}$. ∎

4.4 Comparison

We compare the proposed algorithm with the algorithm in [20]. We denote by PA a point addition in $\mathbb{G}_1$ (or $\mathbb{G}_2$), by SM a point multiplication in $\mathbb{G}_1$ (or $\mathbb{G}_2$), by M a multiplication in $\mathbb{G}_T$, by Inv an inverse in $\mathbb{G}_T$, by Exp an exponentiation in $\mathbb{G}_T$, and P a computation of the bilinear pairing. We omit

---

[5] The computation of bilinear pairings is closely related to the security parameters (that determines the security levels), the kinds of curves (supersingular curves, ordinary curves, or hyperelliptic curves), the kinds of bilinear pairings (the Weil pairing, the Tate pairing, or the Eta pairing), the finite field (the characteristic is 2, 3 or p) and embedding degree *etc.* Koblitz and Menezes [36] presented some examples of the pairings evaluation under the various parameters. For example, it takes roughly $22n$ multiplications in finite filed $\mathbb{GF}(p)$ to compute the Tate pairing $e(A, B)$ when $\mathbb{E}$ is a supersingular elliptic curve defined over $\mathbb{GF}(p)$ with embedding degree $k = 2$, where p is a 512-bit prime in order to achieve 80-bit security level.

other operations such as modular additions in $\mathbb{Z}_q$.

Table 1. Comparison of the two algorithms

	Algorithm [20]	Algorithm **Pair**
T	**10 Exp + 2 Inv + 6 SM** + 4 PA + 6 M	5 PA + 4 M
U	4 P (U)	4 P (U_1) + 4 P (U_2)

Table 1 presents the comparison of the efficiency between algorithm [20] and our proposed algorithm **Pair**. Compared with the algorithm [20], the proposed algorithm **Pair** is much superior in efficiency. More precisely, the outsourcer T does not require the prohibitively expensive operations SM and Exp in our algorithm **Pair** (note that a computationally limited device may be incapable to perform such operations at all). Moreover, the computation of SM (or Exp) is comparable to that of a pairing in some cases, and this will violate the motivation of the outsourcing computations.

On the other hand, it takes the servers U to perform 8P in our algorithm **Pair** (4P for each server U_i). Besides, the computation for *Rand* is about 3P + 3Exp + 9SM, while it is negligible due to the table-lookup method. Therefore, the proposed algorithm **Pair** requires more computation load in the server side compared with [20]. However, note that the server is much more computationally powerful, and thus the efficiency of our algorithm will not be affected in this sense.

5 Secure Outsourcing Algorithms for Identity-based Encryptions and Signatures

In this section, we utilize the proposed subroutine **Pair** to give two secure outsourcing algorithms for Boneh-Franklin identity-based encryption scheme [11] and Cha-Cheon identity-based signature scheme [18], where a special case of bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ is used (i.e., $\mathbb{G}_1 = \mathbb{G}_2$).

Note that the outsourcer T is assumed to be a computationally limited device that cannot carry out the prohibitively expensive computations such as bilinear pairings, point multiplications, modular exponentiations, and so on, thus the proposed two algorithms requires an additional subroutine **SM** [19] for outsourcing the computations of point multiplications in $\mathbb{G}_1$.

5.1 *Outsource-secure Boneh-Franklin Identity-based Encryptions*

The proposed outsource-secure Boneh-Franklin encryption scheme consists of the following efficient algorithms:

- **Setup:** Chooses a random $s \in \mathbb{Z}_q^*$ and sets $P_{pub} = sP$. Define four cryptographic hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1^*$, $H_2 : \mathbb{G}_T \rightarrow \{0,1\}^n$ for some n, $H_3 : \{0,1\}^n \times \{0,1\}^n \rightarrow \mathbb{Z}_q^*$ and $H_4 : \{0,1\}^n \rightarrow \{0,1\}^n$. The public parameters of the system are

$$params = \{\mathbb{G}_1, \mathbb{G}_T, e, q, P, P_{pub}, H_1, H_2, H_3, H_4\}.$$

  The master key is s.

- **Extract:** On input an identity ID, run the extract algorithm to obtain the secret key $S_{ID} = sH_1(ID)$.

- **Encryption:** On input the public key ID and a message $m \in \{0,1\}^n$, the outsourcer T runs the subroutine **Pair** and **SM** to generate the ciphertext C as follows:

(1) T chooses a random $\sigma \in \{0,1\}^n$ and computes $r = H_3(\sigma, m)$.

(2) T runs **SM** to obtain $C_1 = rP$ and $R = rH_1(ID)$.

(3) T runs **Pair** to obtain **Pair**$(R, P_{pub}) \to \varphi$.

(4) T computes $C_2 = \sigma \oplus H_2(\varphi)$ and $C_3 = m \oplus H_4(\sigma)$.

(5) T outputs the ciphertext $C = (C_1, C_2, C_3)$.

- **Decryption:** On input the secret key S_{ID}, and the ciphertext $C = (C_1, C_2, C_3)$, the outsourcer T' runs the subroutine **Pair** and **SM** to compute the message m as follows:

(1) T' runs **Pair** to obtain **Pair**$(S_{ID}, C_1) \to \varphi$.

(2) T' computes $\sigma = C_2 \oplus H_2(\varphi)$.

(3) T' computes $m = C_3 \oplus H_4(\sigma)$.

(4) T' computes $r = H_3(\sigma, m)$ and then runs **SM** to obtain rP.

(5) T' outputs m if and only if $C_1 = rP$.

Remark 3. Note that the outsourcer only needs to perform 6 hash and 4 bitwise operations (instead of 2 pairings and 3 point multiplications) in the above encryption scheme.

5.2 Outsource-secure Cha-Cheon Identity-based Signatures

The proposed outsource-secure Cha-Cheon signature scheme consists of the following efficient algorithms:

- **Setup:** Chooses a random $s \in \mathbb{Z}_q^*$ and sets $P_{pub} = sP$. Define two cryptographic hash functions $H_1 : \{0,1\}^* \times \mathbb{G}_1 \to \mathbb{Z}_q$, $H_2 : \{0,1\}^* \to \mathbb{G}_1$. The public parameters of the system are $params = \{\mathbb{G}_1, \mathbb{G}_T, e, q, P, P_{pub}, H_1, H_2\}$. The master key is s.
- **Extract:** On input an identity ID, run the extract algorithm to obtain the signing key $S_{ID} = sH_2(ID)$.
- **Sign:** On input the singing key S_{ID} and a message m, the outsourcer T runs the subroutine **SM** to generate the signature σ as follows:

(1) T chooses a random $r \in \mathbb{Z}_q^*$ and runs **SM** to obtain $U = rH_2(ID)$.

(2) T computes $h = H_1(m, U)$.

(3) T runs **SM** to obtain $V = (r + h)S_{ID}$. The signature is $\sigma = (U, V)$.

- **Verify:** On input the verification key ID, the message m, and the signature $\sigma = (U, V)$, the outsourcer T' runs the subroutine **Pair** and **SM** to verify the signature σ as follows:

(1) T' computes $h = H_1(m, U)$.

(2) T' runs **SM** to obtain $hH_2(ID)$ and computes $T = U + hH_2(ID)$.

(3) T' runs **Pair** to obtain $\mathbf{Pair}(P, V) \to \beta_1$ and $\mathbf{Pair}(P_{pub}, T) \to \beta_2$.

(4) T' outputs 1 if and only if $\beta_1 = \beta_2$.

Remark 4. Note that the outsourcer only needs to perform 2 hash and 1 point addition operations (instead of 2 pairings and 3 point multiplications) in the above signature scheme.

6 Conclusions

In this paper, we first proposed an efficient and secure outsourcing algorithm for bilinear pairings in the two untrusted program model. A distinguishing property of our proposed algorithm is that the (resources-limited) outsourcer never requires to accomplish some expensive operations such as point multiplications and exponentiations.

The security model of our outsourcing algorithm requires the outsourcer to interact with *two* untrusted while non-colluding cloud servers (the same as [32]). Therefore, an interesting open problem is whether there is an efficient algorithm for securely outsourcing bilinear pairings using only one untrusted cloud server.

Acknowledgements

We are grateful to the anonymous referees for their invaluable suggestions. This work is supported by the National Natural Science Foundation of China (Nos. 61272455 and 61100224), China 111 Project (No. B08038), Doctoral Fund of Ministry of Education of China (No.20130203110004), Program for New Century Excellent Talents in University (No. NCET-13-0946), and the Fundamental Research Funds for the Central Universities (No. BDY151402). The second author is supported by the Australian Reserach Council Future Fellowship (FT0991397) and also partly funded by the Australian Research Council Discovery Project DP130101383.

References

[1] Atallah M.J., Frikken K.B.: Securely outsourcing linear algebra computations. Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS). pp. 48-59 (2010).

[2] Abadi M., Feigenbaum J., Kilian J.: On hiding information from an oracle. Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC). pp. 195-203 (1987).

[3] Atallah M.J., Pantazopoulos K.N., Rice J.R., Spafford E.H.: Secure outsourcing of scientific computations. Advances in Computers. vol.54, pp. 216-272 (2001).

[4] Atallah M.J., Li J.: Secure outsourcing of sequence comparisons. International Journal of Information Security, **4**(4), 277-287 (2005).

[5] Benjamin D., Atallah M.J.: Private and cheating-free outsourcing of algebraic computations. Proceeding of the 6th Annual Conference on Privacy, Security and Trust (PST). pp. 240-245 (2008).

[6] Blanton M., Aliasgari M.: Secure Outsourcing of DNA Searching via Finite Automata. Data and Applications Security and Privacy XXIV, LNCS 6166, Springer-Verlag, pp. 49-64 (2010).

[7] Blanton M., Aliasgari M.: Secure outsourced computation of iris matching. Journal of Computer Security, **20**(2-3), 259-305 (2012).

[8] Blanton M.: Improved conditional e-payments. ACNS 2008. LNCS 5037, Springer-Verlag, pp. 188-206 (2008).

[9] Blanton M., Atallah M.J., Frikken K.B., Malluhi Q.: Secure and efficient outsourcing of sequence comparisons. ESORICS 2012. LNCS 7459, pp. 505-522 (2012).

[10] Beuchat J., González-Díaz J.E., Mitsunari S., Okamoto E., Rodríguez-Henríquez F., and Teruya T.: High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves, Pairing 2010. LNCS 6487, pp. 21-39 (2010).

[11] Boneh D., Franklin M.: Identity-based encryption from the Weil pairings. Advances in Cryptology-Crypto 2001. LNCS 2139, pp. 213-229 (2001).

[12] Benabbas S., Gennaro R., Vahlis Y.: Verifiable delegation of computation over large datasets. Advances in Cryptology-Crypto 2011. LNCS 6841, pp. 111-131 (2011).

[13] Barreto P., Galbraith S., Ó' hÉigeartaigh C., Scott M.: Efficient pairing computation on supersingular Abelian varieties. Designs, Codes and Cryptography, **42**(3), 239-271 (2007).

[14] Blum M., Luby M., Rubinfeld R.: Self-testing/correcting with applications to numerical problems. Journal of Computer and System Science, **47**(3), 549-595 (1993).

[15] Boneh D., Lynn B., Shacham H.: Short signatures from the Weil pairings. Advances in Cryptology-Asiacrypt 2001. LNCS 2248, pp. 514-532 (2001).

[16] Boyko V., Peinado M., Venkatesan R.: Speeding up discrete log and factoring based schemes via precomputations. Advances in Cryptology-Eurocrypt 1998. LNCS 1403, pp.221-232 (1998).

[17] Chow S., Au M., Susilo W.: Server-aided signatures verification secure against collusion attack. Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS). pp. 401-405 (2011).

[18] Cha J., Cheon J.H.: An identity-based signature from gap Diffie-Hellman groups. Public Key Cryptography-PKC 2003. LNCS 2567, pp. 18-30 (2003).

[19] Chen X., Li J., Ma J., Tang Q., Lou W.: New algorithms for secure outsourcing of modular exponentiations. ESORICS 2012. LNCS 7459, pp. 541-556 (2012).

[20] Chevallier-Mames B., Coron J., McCullagh N., Naccache D., Scott M.: Secure delegation of elliptic-curve pairing. CARDIS 2010. LNCS 6035, pp. 24-35 (2010).

[21] Chaum D., Pedersen T.: Wallet databases with observers. Advances in Cryptology-Crypto 1992. LNCS 740, pp. 89-105 (1993).

[22] Canetti R., Riva B., Rothblum G.: Practical delegation of computation using multiple servers. Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS). pp. 445-454 (2011).

[23] Carbunar B., Tripunitara M.: Conditioal payments for computing markets. CANS 2008. LNCS 5339, pp. 317-331 (2008).

[24] Carbunar B., Tripunitara M.: Fair payments for outsourced computations. SECON 2010. pp. 529-537 (2010).

[25] Galbraith S., Paterson K., Smart N.: Pairings for cryptographers. Discrete Applied Mathematics, **156**(16), 3113-3121 (2008).

[26] Green M., Hohenberger S., Waters B.: Outsourcing the decryption of ABE ciphertexts. Proceedings of the 20th USENIX conference on Security. The full version can be found at http://static.usenix.org/events/sec11/tech/full-papers/Green.pdf (2011).

[27] Gennaro R., Gentry C., Parno B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. Advances in Cryptology-Crypto 2010. LNCS 6223, pp. 465-482 (2010).

[28] Goldwasser S., Kalai Y.T., Rothblum G.N.: Delegating computation: interactive proofs for muggles. Proceedings of the ACM Symposium on the Theory of Computing (STOC). pp. 113-122 (2008).

[29] Girault M., Lefranc D.: Server-aided verification: theory and practice. Advances in Cryptology-ASIACRYPT 2005. LNCS 3788, pp. 605-623 (2005).

[30] Goldwasser S., Micali S., Rackoff C.: The knowledge complexity of interactive proof-systems. SIAM Journal on Computing, **18**(1), 186-208 (1989).

[31] Golle P., Mironov I.: Uncheatable distributed computations. CT-RSA 2001. LNCS 2020, pp. 425-440 (2001).

[32] Hohenberger S., Lysyanskaya A.: How to securely outsource cryptographic computations. TCC 2005. LNCS 3378, pp. 264-282. The full version can be found at http://www.cs.jhu.edu/ susan/papers/HL05.pdf (2005).

[33] Hess F., Smart N., Vercauteren F.: The Eta pairing revisited. IEEE Transactions on Information Theory, **52**(10), 4595-4602 (2006).

[34] Joux A.: A one round protocol for tripartite Diffie-Hellman. Algorithmic Number Theory Symposium-ANTS IV. LNCS 1838, pp. 385-394 (2000).

[35] Kang B., Lee M., Park J.: Efficient Delegation of pairing computation. Cryptology ePrint Archive, Report 2005/259 (2005).

[36] Koblitz N., Menezes A.: Pairing-based cryptography at high security levels. Cryptography and Coding 2005. LNCS 3796, pp. 13-36 (2005).

[37] Kilian J.: Improved efficient arguments (preliminary version). Advances in Cryptology-Crypto 1995. pp. 311-324. (1995).

[38] Micali S.: CS proofs. Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS). pp. 436-453 (1994).

[39] Matsumoto T., Kato K., Imai H.: Speeding up secret computations with insecure auxiliary devices. Advances in Cryptology-Crypto 1988. LNCS 403, pp. 497-506, (1988).

[40] Parno B., Raykova M., Vaikuntanathan V.: How to delegate and verify in public: verifiable computation from attribute-based encryption. TCC 2012. LNCS 7194, pp. 422-439 (2012).

[41] Schnorr C.P.: Efficient signature generation for smart cards. Journal of Cryptology, **4**(3), 239-252 (1991).

[42] Scott M., Costigan N., Abdulwahab W.: Implementing cryptographic pairings on smartcards. CHES 2006. LNCS 4249, pp. 134-147 (2006).

[43] Shi L., Carbunar B., Sion R.: Conditional E-cash. FC 2007. LNCS 4886, pp. 15-28 (2007).

[44] Tsang P., Chow S., Smith S.: Batch pairing delegation. IWSEC, pp. 74-90 (2007).

[45] Wang C., Ren K., Wang J.: Secure and practical outsourcing of linear programming in cloud computing. Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM). pp. 820-828, (2011).