Secure Bilinear Pairing Outsourcing Made More Efficient and Flexible

Haibo Tian*
[1]School of Information Science and Technology,Sun Yat-sen University
[2]Guangdong Provincial Key Laboratory of Information Security, Guangzhou 510006, China
tianhb@mail.sysu.edu.cn

Fangguo Zhang
[1]School of Information Science and Technology, Sun Yat-sen University
[2]Guangdong Provincial Key Laboratory of Information Security, Guangzhou 510006, China
[3]Department of Computer Science and Engineering University at Buffalo, State University of New York, NY 14260-2500, U.S.A.
isszhfg@mail.sysu.edu.cn

Kui Ren
Department of Computer Science and Engineering University at Buffalo, State University of New York, NY 14260-2500, U.S.A.
kuiren@buffalo.edu

ABSTRACT

The increasing availability of cloud computing allows more and more mobile devices to outsource expensive computations. Among these computations, bilinear pairing is very fundamental and frequently-used by many modern cryptographic protocols. Currently, the most efficient outsourcing algorithm of bilinear pairings requires about 5 point additions in $\mathbb{G}_1$ and $\mathbb{G}_2$ and 4 multiplications in $\mathbb{G}_T$ under the one-malicious version of a two-untrusted-program assumption. And the result of the algorithm is checkable with a probability about 1/2. In this paper, we improve the state-of-the-art by proposing two new outsourcing algorithms for bilinear pairings. One is a more efficient outsourcing algorithm under the same assumption with the same checkability. The other is more flexible under a two-untrusted-program assumption with improved checkability. Both algorithms are better suited to various applications where on-line computations are strictly limited due to the lack of available computing resources.

Categories and Subject Descriptors

F.2.1 [**Numerical Algorithms and Problems**]: Computations in finite fields; E.3 [**DATA ENCRYPTION**]: Public key cryptosystems

General Terms

Privacy

*The Corresponding Author

ASIA CCS'15, April 14–17, 2015, Singapore.

http://dx.doi.org/10.1145/2714576.2714615.

Keywords

Secure Outsourcing; Bilinear Pairings; Efficiency; Checkability

1. INTRODUCTION

The development of cloud computing enables people to access almost unlimited computing resources in an on-demand fashion. People with mobile devices or other energy-limited devices may benefit from this pattern to fulfill some complex computations that are unaffordable in their devices alone, such as linear programming [27] and convex optimization [32]. In the setting of cryptographic applications, it is satisfactory if a device could outsource some energy-consuming operations to a cloud, and keep the security properties of the applications unchanged [22]. As a basic operation in pairing-based cryptographic applications, the bilinear pairing evaluation attracts researchers to study suitable security model and efficient implementations for outsourcing.

There are mainly three aspects about the security model.

- The first is about secrecy. Some sensitive information involved in an outsourced process should be kept secrete. The requirement comes from a semi-trusted cloud server assumption where a cloud server may be curious to peek client's data. In practice, cloud servers are usually owned by some corporations that may be not trusted by clients. So this assumption is sound.

- The second is about checkability. The result of an outsourced computation should be checkable at least with some probability. This requirement again comes from the semi-trusted cloud server assumption. A service program in a could server may have bugs or malicious functions. It is also attractive for a cloud server to finish a computing task as soon as possible, which may lead to incomplete computation or even zero computation where simply a wrong result is returned.

- The last aspect is about assumptions on the number and trustability of programs to implement an algorith-

m. A one-untrusted-program (OUP) assumption allows only one program to implement an algorithm, and the program could be malicious. A one-malicious version of a two-untrusted-program (OMTUP) assumption allows two programs to implement an algorithm, and only one of them could be malicious. A two-untrusted-program (TUP) assumption allows two programs to implement an algorithm and both of them could be malicious.

In practice, the OUP assumption needs a client to access only one program to outsource its computation task. An algorithm under the OUP assumption usually needs more computations than under other assumptions. Most outsourcing algorithms for bilinear pairings are under the OUP assumption. The OMTUP and TUP assumptions need a client to access two programs. Under the OMTUP assumption, the two programs are expected to be produced by different vendors. Under the TUP assumption, the two programs may be from the same vendor.

The efficiency of an algorithm with outsourced computation is a basic goal and is the reason of the existence of such an algorithm. Considering bilinear pairings, before the work of Chen et al. [12], all outsourcing algorithms [16, 13, 8] need scalar multiplications and exponentiations. The fast implementations of bilinear pairings on energy-limited devices [24, 25] almost make the effort of pairings outsourcing useless. Just for some parameter sets, there are some benefits to outsource bilinear pairings. The outsourcing algorithm in [12] only needs a few point additions in $\mathbb{G}_1$ and $\mathbb{G}_2$ and multiplications in $\mathbb{G}_T$, which makes their work valuable.

Since a bilinear pairing is a basic operation in pairing based applications, a pairing outsourcing algorithm has a wide usage. Chen et al. [12] gave two examples. One is the well-known Boneh-Franlin identity-based encryption scheme [4]. The other is the Cha-Cheon identity-based signature scheme [14]. Both schemes need on-line pairing evaluations. Their results show that the on-line computations are largely reduced.

We notice the similarity of on-line/off-line computation algorithms and the outsourcing algorithms in [17, 12, 28, 8]. They all need some off-line computations to support the on-line computations. The difference is whether the on-line computations are outsourced. From this standpoint, we may view the mentioned algorithms as extensions to on-line/off-line computation algorithms to save the on-line computing resources. So we claim the algorithms are suitable to applications where on-line computations have strict limitations on available computing resources. In practice, the off-line computations may be finished when a mobile device is charging or simply be finished by another device that puts the off-line computation results to the mobile device.

This paper mainly proposes two outsourcing algorithms. One is a more efficient bilinear pairing outsourcing algorithm in the framework of [12]. The other is an algorithm in a more flexible framework with a better checkability. We also give two pre-computation algorithms for the two outsourcing algorithms respectively that are based on a well-known EBPV algorithm.

1.1 Related Works

The outsourcing computation of cryptographic operations is a technique that has been studied for a long time [19, 9, 18]. There are also some generic constructions [33, 7] for secure outsourcing computation. For the special case of bilinear pairing outsourcing, the closely related works are as follows.

Girault and Lefranc [16] proposed some protocols for pairing-based schemes for the first time. They suggest a method to blind a point. Suppose two points $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$. A client wants to compute $e(A, B)$ and to hide the point B from a server. The client could select a random value $u \in \mathbb{Z}_q$ where q is the order of $\mathbb{G}_1$ and $\mathbb{G}_2$. And the client sends A and uB to the server. After the server returns $e(A, uB)$, the client could recover $e(A, B)$.

Chevallier-Mames et al. [2, 13] proposed a protocol for secure delegation of the elliptic-curve pairing. The secure delegation in their proposal means that both points could be hidden from a server, and that the server's computation result is checkable. Basically, they make a server compute two comparable results to check the correctness of a pairing evaluation. Kang et al. [3] proposed a simpler protocol to improve the efficiency of the protocol in [2]. They feed a server with a set of new points and obtain two comparable results with a smaller cost.

Tsang et al. [21] proposed the concept of batch pairing delegation, and constructed several protocols for different types of bilinear pairings. They showed that the batch pairing delegation was more efficient than multiple runs of non-batch delegation protocols at that time.

Wu et al. [30, 31] focused on server-aided signature verification. They specially study the verification of Boneh-Lynn-Shacham (BLS) signature [5]. Their final construction shows a similar idea to exploit two comparable results returned by a server to check the correctness of server's computation. Wang et al. [29] proposed a new security model in the setting of server-aided verification. Chow et al. [15] proposed another new model in the same setting, and proposed a method to exploit pairing delegation protocols [21] for signature verification.

Canard et al. [8] proposed a new method to delegate a pairing evaluation. Their construction shows again the similar idea to use two comparable results returned by a server to check the correctness of server's computation. They contribute a new efficient construction to implement such an idea. Their construction for public points outsourcing needs two scalar multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$, one exponentiation in $\mathbb{G}_T$ and a membership test in $\mathbb{G}_T$. They noticed the work of Ana and Francisco [25] where an optimal-Ate pairing over Barreto-Naehrig curves [1] was implemented. They then argue that the benefit of pairings outsourcing may be the save of area to implement a pairing evaluation. In fact, Scott et al. [24] have showed an implementation of cryptographic pairings on smart cards, where the Section 7 shows that a pairing outsourcing algorithm is only meaningful for Ate pairings.

Surprisingly, Chen et al. [12] proposed an efficient algorithm for secure outsourcing of bilinear pairings. They only require a client to compute 5 point additions in $\mathbb{G}_1$ and $\mathbb{G}_2$ and 4 multiplications in $\mathbb{G}_T$. No scalar multiplications or exponentiations are needed.

1.2 Contributions

More Efficient Algorithm: We exploit a trick in [24, 8] to use an inverse of a random value in a pre-computation phase. Basically, suppose $e(A, B)$ is expected to be out-

sourced. We let a server compute $e(A+x_1P_1, B+x_1^{-1}x_2P_2)$ where P_1 and P_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Note $e(x_1P_1, x_1^{-1}x_2P_2) = e(P_1, P_2)^{x_2}$. Then some computations in [12] could be saved. We also observe that some computations in [12] could be further pre-computed. For example, the second query to the program U_1 in [12] is not related to the inputs (A, B). That computation could be pre-computed, and the queries to U_1 could be reduced.

Pre-computation Algorithms: The outsourcing algorithm in [12] needs precomputations of three six-tuples each of which includes three scalar multiplications and one pairing evaluation. We propose two algorithms based on a well-known EBPV algorithm [20] for precomputations. We further exploit an idea in [28] to build a dynamic table where each entry is used to outsource a pairing evaluation. Our efficient algorithm needs ten random elements for one pairing evaluation. The other algorithm needs twelve random elements for one pairing evaluation.

Two Untrusted Programs: We notice that an algorithm under the OUP assumption may need more computations than under the OMTUP and TUP assumptions. And the OMTUP assumption may be not well-suited to some practical deployment of cloud services. For example, many cloud services may be based on the same platform provided by a vendor. We then propose a flexible algorithm under the TUP assumption. We let two programs compute two closely related outputs whose relationship could be checked by a client with a small cost, similar to [2, 30, 8]. We emphasize that an algorithm secure under the OMTUP assumption may have different behaviors under the TUP assumption. For example, our efficient algorithm and an algorithm in [10, 11] exploit the OMTUP assumption to check a result by comparing two values returned by two programs while a client does not know a correct answer. This trick is invalid under the TUP assumption.

Improvement on Checkability: The outsourcing algorithm under the TUP assumption has a probability $(1 - 1/3s)^2$ to check the result for a security parameter s. Our method gives a client a choice to improve the checkability at the cost of several computations. Note that in some applications of outsourcing algorithms, the checkability may be crucial. For example, if a signature verification procedure employs a bilinear pairing outsourcing algorithm, the checkability of the outsourcing algorithm determines the correctness of the signature verification procedure. Chow et al. [15] discussed this problem in their security analysis section.

1.3 Organizations

The next section shows the pre-computation algorithms based on EBPV. Section 3 is a more efficient outsourcing algorithm for bilinear pairings with the descriptions of security model and proofs. The algorithm under the TUP assumption and its proofs are in Section 4. Section 5 includes a comparison. The last section concludes the paper.

2. PRE-COMPUTATION ALGORITHMS

2.1 EBPV

Boyko, Peinado and Venkatesan [6] proposed a method to speed up discrete log and factoring based schemes via precomputations. It is called as BPV algorithm. Nguyen, Shparlinski and Stern [20] showed an extension of the BPV algorithm called as EBPV algorithm. We here review the EBPV algorithm in [20] as follows.

Let g be a generator of a cyclic multiplicative group $\mathbb{Z}_p^*$ with order q where p and q are two large primes.

- *Preprocessing Step*: Generate n random integers α_1, ..., $\alpha_n \in \mathbb{Z}_q$. For $j = 1, \ldots, n$, compute $\beta_j = g^{\alpha_j} \bmod p$, and store the values of α_j and β_j in a table.
- *Pair Generation*: When a pair (x, g^x) is needed, randomly generate $S \in \{1, \ldots, n\}$ such that $|S| = k$. For each $j \in S$, randomly select $\chi_j \in \{1, \ldots, h-1\}$ where $h > 1$ is a small integer. Compute

$$x = \sum_{j \in S} \alpha_j \chi_j \bmod q \tag{1}$$

and

$$X = \prod_{j \in S} \beta_j^{\chi_j} \bmod p. \tag{2}$$

If $x = 0 \bmod q$, start again. Otherwise return the pair (x, X).

Note that Nguyen et al. [20] have studied the distribution of modular sums and have shown that by choosing a suitable parameter set, the statistical distance between the distribution of a modular sum and the uniform distribution could be arbitrarily small.

2.2 Bilinear Pairings

In the setting of bilinear pairings, we use the following symbols. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic additive groups generated by P_1 and P_2 respectively. The order of $\mathbb{G}_1$ and $\mathbb{G}_2$ is a large prime and also denoted by the symbol q. Define $\mathbb{G}_T$ to be a cyclic multiplicative group of the same order q. A bilinear pairing is defined as a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

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

Since the EBPV algorithm has no limitations on the cyclic groups, it could be trivially applied to the setting of bilinear pairings.

2.3 Dynamic Table

Wang et al. [28] proposed a BPV$^+$ algorithm employing two tables. One table is a static table to store precomputations. The other is a dynamic table that maintains some pairs of elements produced by a BPV algorithm [6]. On each invocation of their BPV$^+$ algorithm, an entry in the dynamic table is used and removed. The dynamic table is expected to be replenished with some fresh random pairs in an idle time of a device. Obviously, this method could be used together with the EBPV algorithm smoothly.

2.4 *RandA*

We next give the first pre-computation algorithm *RandA*. It prepares random vectors for our more efficient algorithm with outsourced computation. The subscripts of random

values are kept the same as they in the corresponding outsourcing algorithm. It includes a static table ST and a dynamic table DT.

- *Preprocessing Step*: Generate n random integers $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_q$. For $j = 1, \ldots, n$ compute $\beta_{j1} = \alpha_j P_1$ and $\beta_{j2} = \alpha_j P_2$, and store the values of α_j, β_{j1} and β_{j2} in a static table ST. Compute $e(P_1, P_2)$ and store the value in ST.

- *Vector Generation*: When a table DT needs a new entry, it is produced as follows. Randomly generate $S \in \{1, \ldots, n\}$ such that $|S| = k$. For each $j \in S$, randomly select $\chi_j \in \{1, \ldots, h-1\}$ where $h > 1$ is a small integer. Compute

$$x_1 = \sum_{j \in S} \alpha_j \chi_j \bmod q \tag{3}$$

If $x_1 = 0 \bmod q$, start again. Else compute

$$x_1 P_1 = \sum_{j \in S} \chi_j \beta_{j1}. \tag{4}$$

Following the above procedure, compute similarly the vectors (x_3, x_3P_1), (x_4, x_4P_2), (x_7, x_7P_1) and (x_8, x_8P_2). Then randomly select $x_2, x_5, x_6 \in \mathbb{Z}_q^*$ and compute

$$\begin{array}{l} 1.\ x_1^{-1}x_2P_2, \\ 2.\ x_1x_2^{-1}x_5P_1, \\ 3.\ x_1^{-1}x_6P_2, \\ 4.\ e(P_1,P_2)^{x_7x_8}, \\ 5.\ e(P_1,P_2)^{x_5+x_6-x_2}. \end{array} \tag{5}$$

The entry

$$\begin{array}{l} (x_1P_1, x_3P_1, x_1x_2^{-1}x_5P_1, x_7P_1, \\ x_1^{-1}x_2P_2, x_4P_2, x_1^{-1}x_6P_2, x_8P_2, \\ e(P_1,P_2)^{x_7x_8}, e(P_1,P_2)^{x_5+x_6-x_2}). \end{array} \tag{6}$$

are stored in the DT table. On each invocation of *RandA*, an entry is returned and removed from DT. And the DT is replenished with fresh random values in an idle time of a device.

2.5 *RandB*

Our flexible algorithm under the TUP assumption needs the following *RandB* algorithm to prepare random vectors. The subscripts of random values are also kept the same as they in the corresponding outsourcing algorithm.

- *Preprocessing Step*: The table ST is filled in the same way as that in the *RandA* algorithm.

- *Vector Generation*: When a table DT needs a new entry, it is produced as follows. A pair (x_1, x_1P_1) is fast sampled in the same way as the *RandA* algorithm. Then randomly select $x_2, x_3, x_4 \in \mathbb{Z}_q^*$, and compute

$$\begin{array}{l} 1.\ x_1x_2^{-1}x_3P_1, \\ 2.\ x_1^{-1}x_2P_2, \\ 3.\ x_1^{-1}x_4P_2, \\ 4.\ (e(P_1,P_2)^{x_3+x_4-x_2}, \end{array} \tag{7}$$

The entry

$$\begin{array}{l} (x_1P_1, x_1x_2^{-1}x_3P_1, \\ x_1^{-1}x_2P_2, x_1^{-1}x_4P_2 \\ e(P_1,P_2)^{x_3+x_4-x_2}). \end{array} \tag{8}$$

are stored in the DT table. On each invocation of *RandB*, an entry is returned and removed from DT. And the DT is replenished with fresh random values in an idle time of a device.

Table 1: Comparison of Precomputations

	Scheme [12]	Algorithm A	Algorithm B
SM	9	3	6
ME	0	2	2
PE	3	0	0
PA	-	$5(k+h-3)$	$2(k+h-3)$

2.6 Comparisons

We show the precomputations needed for a pairing evaluation in our two algorithms and that in Chen et al.'s algorithm [12]. We name our efficient algorithm in Section 3 as "Algorithm A". The algorithm with outsourced computation in Section 4 is called as "Algorithm B". We then note that for a outsourced pairing evaluation, the *RandA* algorithm is called once in Algorithm A, and the *RandB* algorithm is called twice in Algorithm B, and Chen et al. need three times six-tuple in their algorithm. We use SM to denote a scalar multiplication, ME a modular exponentiation, PE a pairing evaluation and PA a point addition. The numbers of SM and PA include all operations in $\mathbb{G}_1$ and $\mathbb{G}_2$. The value k is the size of a set S in the algorithms *RandA* and *RandB*. According to the parameter sets in [20], the value k is about twenties and the value h is less than 10. It is also noted in [20] that the computation cost of the value X is about $k + h - 3$ group operations. The comparison result is in Table 1. It is obvious that *RandA* and *RandB* algorithms need less computations. Note that the computations in *Preprocessing Step* are not included since they are produced once and used many times.

3. AN EFFICIENT ALGORITHM

This section includes the construction of our "Algorithm A" and its related security model and proofs.

3.1 Construction of Algorithm A

We use $U_i(R, Q) \rightarrow e(R, Q)$, $i \in \{1, 2\}$, to denote the party U_i taking (R, Q) as inputs and producing $e(R, Q)$ as an output. And we use T to denote a trusted device with limited computation resources. System parameters include $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, q, P_1, P_2)$ that are defined in Section 2.2. The inputs of algorithm A are points $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$. The output of algorithm A is expected to be $e(A, B)$. The algorithm is executed as follows.

- *Init*: T calls *RandA* to fetch random values

$$\begin{array}{l} (x_1P_1, x_3P_1, x_1x_2^{-1}x_5P_1, x_7P_1, \\ x_1^{-1}x_2P_2, x_4P_2, x_1^{-1}x_6P_2, x_8P_2, \\ e(P_1,P_2)^{x_7x_8}, e(P_1,P_2)^{x_5+x_6-x_2}). \end{array} \tag{9}$$

- *Computation*: T queries U_1 in random orders

  1. $U_1(A + x_1P_1, B + x_1^{-1}x_2P_2) \rightarrow \alpha_1$;
  2. $U_1(x_3P_1, x_4P_2) \rightarrow \alpha_2$;

  Similarly, T queries U_2 in random orders

  1. $U_2(A + x_1x_2^{-1}x_5P_1, -x_1^{-1}x_2P_2) \rightarrow \alpha_1'$;

2. $U_2(-x_1P_1, B + x_1^{-1}x_6P_2) \to \alpha_2'$;
3. $U_2(x_3P_1, x_4P_2) \to \alpha_3'$;
4. $U_2(x_7P_1, x_8P_2) \to \alpha_4'$;

- *Recover*: T checks whether

$$\alpha_2 = \alpha_3'$$

and

$$e(P_1, P_2)^{x_7x_8} = \alpha_4'.$$

If the equations hold, it computes

$$o = \alpha_1\alpha_1'\alpha_2'e(P_1, P_2)^{x_5+x_6-x_2}$$

and produces o as an output. Otherwise, it rejects and produces "Error".

3.2 Security Model

Chen et al. [12] proved their algorithm in a security model proposed in [17]. We introduce the model as follows.

An outsourcing model includes three players, a trusted party T, an untrusted party U and an untrusted environment E. The party T has limited computation resource and tries to outsource its computation task to the party U. The party U may be a program written by E and installed on a computing device. The party E produces the program U and interacts indirectly with U through T. The goal of E and U is to learn something interesting about T. The goal of T is to implement an algorithm Alg by interacting with U. An outsourcing implementation of an algorithm is secure if U and E could learn nothing interesting about T's inputs and outputs.

Next is the formal definition of "an algorithm with outsource I/O" [17, 12].

Definition 1. An algorithm with outsource I/O takes five inputs and produces three outputs which are divided to secret, protected and unprotected categories according to the knowledge of U and E. An input or output is secret if neither E nor U knows it. If only E knows an input or output, it is called as protected. Otherwise, an input or output is unprotected. An honest party produces the first three inputs known as honest secret, honest protected, and honest unprotected inputs. The environment E produces the following adversarial protected, and adversarial unprotected inputs. The algorithm produces secret, protected and unprotected outputs.

The security definition of an algorithm with outsource I/O is mainly to ensure that the party E knows nothing about the secret input and output of an implementation T^U even if the party U is produced by the party E.

Definition 2. Let Alg be an algorithm with outsource I/O. An implementation T^U of Alg is secure if:

1. T^U is a correct implementation of Alg;
2. For all probabilistic polynomial time adversaries E and U, there are probabilistic expected polynomial time simulators (S_1, S_2) such that the following pairs of random variables are computationally indistinguishable.
   - Pair One: $EVIEW_{real} \sim EVIEW_{ideal}$ that are views of the party E participating two processes.
     - $EVIEW_{real} = EVIEW_{real}^i$ if $stop^i = TRUE$ where

$EVIEW_{real}^i = (estate^i, y_p^i, y_u^i)$ and
$(istate^i, x_{hs}^i, x_{hp}^i, x_{hu}^i) \leftarrow I(1^\kappa, istate^{i-1})$;
$(estate^i, j^i, x_{ap}^i, x_{au}^i, stop^i) \leftarrow$
$E(1^\kappa, EVIEW_{real}^{i-1}, x_{hp}^i, x_{hu}^i)$;
$(tstate^i, ustate^i, y_s^i, y_p^i, y_u^i) \leftarrow$
$T^{U(ustate^{i-1})}(tstate^{i-1}, x_{hs}^{j^i}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^i, x_{au}^i)$.

     The algorithm I is honest to take a security parameter κ and its $i-1$ round internal states $istate^{i-1}$ and to produce its i round honest state and honest inputs $(x_{hs}^i, x_{hp}^i, x_{hu}^i)$ for T^U. The adversary E takes κ, x_{hp}^i, x_{hu}^i and its $i-1$ round view $EVIEW_{real}^{i-1}$ as inputs to produce its i round internal state $estate^i$, the order of honest inputs j^i, the i round adversarial inputs x_{ap}^i and x_{au}^i, and a terminal signal $stop^i$. The adversary U takes its $i-1$ round internal state $ustate^{i-1}$ to interact with T in the round i interaction. The implementation T^U takes the five inputs and the $i-1$ round internal state $tstate^{i-1}$ of T to produce the round i internal states of T and U, and the three round i outputs. All states are initially empty.
     - $EVIEW_{ideal} = EVIEW_{ideal}^i$ if $stop^i = TRUE$ where

$EVIEW_{ideal}^i = (estate^i, z_p^i, z_u^i)$ and
$(istate^i, x_{hs}^i, x_{hp}^i, x_{hu}^i) \leftarrow I(1^\kappa, istate^{i-1})$;
$(estate^i, j^i, x_{ap}^i, x_{au}^i, stop^i) \leftarrow$
$E(1^\kappa, EVIEW_{ideal}^{i-1}, x_{hp}^i, x_{hu}^i)$;
$(astate^i, y_s^i, y_p^i, y_u^i) \leftarrow$
$Alg(astate^{i-1}, x_{hs}^{j^i}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^i, x_{au}^i)$;
$(sstate^i, ustate^i, Y_p^i, Y_u^i, rep^i) \leftarrow$
$S_1^{U(ustate^{i-1})}(sstate^{i-1}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^i, x_{au}^i,$
$y_p^i, y_u^i)$;
$(z_p^i, z_u^i) = rep^i(Y_p^i, Y_u^i) + (1 - rep^i)(y_p^i, y_u^i)$.

     The algorithms I, E and U are similar to those in the real process. The algorithm Alg takes the five inputs and the $i-1$ round internal state $astate^{i-1}$ of Alg to produce the three outputs and the round i state. The simulated implementation S_1^U takes its $i-1$ round internal state $sstate^{i-1}$ and all the protected and unprotected inputs and outputs to produce the round i internal states of S_1 and U, the simulated protected and unprotected output Y_p^i and Y_u^i and a response signal $rep^i \in \{0, 1\}$. The response signal is used to determine the round i outputs z_p^i and z_u^i for $EVIEW_{ideal}^i$.
   - Pair Two: $UVIEW_{real} \sim UVIEW_{ideal}$ that are views of the party U participating two processes.
     - $UVIEW_{real} = UVIEW_{real}^i$ if $stop^i = TRUE$ where $UVIEW_{real}^i = (ustate^i)$ and the $ustate^i$ is defined in the $EVIEW_{real}^i$ of the above "Pair One" definition.
     - $UVIEW_{ideal} = UVIEW_{ideal}^i$ if $stop^i = TRUE$

where

$UVIEW^i_{ideal} = ustate^i$ and
$(istate^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^\kappa, istate^{i-1})$;
$(estate^i, j^i, x^i_{ap}, x^i_{au}, stop^i) \leftarrow$
$E(1^\kappa, estate^{i-1}, x^i_{hp}, x^i_{hu}, y^{i-1}_p, y^{i-1}_u)$;
$(astate^i, y^i_s, y^i_p, y^i_u) \leftarrow$
$Alg(astate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$;
$(sstate^i, ustate^i) \leftarrow$
$S_2^{U(ustate^{i-1})}(sstate^{i-1}, x^{j^i}_{hu}, x^i_{au})$.

The algorithms I and E are the same as those in the $EVIEW^i_{real}$ of the above "Pair One" definition. The algorithm Alg is defined in the same way as that in the $EVIEW^i_{ideal}$ of the above "Pair One" definition. The simulated implementation S_2^U takes its round $i-1$ state and unprotected inputs to produce two round i states for the simulator S_2 and U.

An algorithm with outsource I/O should be more efficient than the algorithm runs in a single device. This is formally defined as "α-Efficient Secure Outsourcing".

Definition 3. A pair of algorithms (T, U) is α-efficient secure outsourcing if T^U is a secure implementation of the algorithm Alg, and if for any input x, the running time of T is no more than an α-multiplicative factor of the running time of Alg.

Since the computation is outsourced to an untrusted party U, it is crucial to check the correctness of the result. This requirement is formally defined as "β-Checkable Secure Outsourcing".

Definition 4. A pair of algorithms (T, U) is β-checkable secure outsourcing if T^U is a secure implementation of the algorithm Alg, and if for any input x, T could detect any deviations of U from U's advertised functionality during the execution of T^U with a probability no less than β.

In summary, an efficient, checkable and secure outsourcing is defined as "(α, β)-Outsource Security".

Definition 5. A pair of algorithms (T, U) has the (α, β)-outsource security property if it is α-efficient and β-checkable secure outsourcing.

According to the number and trustability of the party U, the OMTUP assumption is defined as follows:

Definition 6. If the party U denotes two noninteractive programs (U_1, U_2), and if only one of them U_i, $i \in_R \{1, 2\}$, is malicious, the algorithms (T, U) are implemented under an OMTUP assumption.

3.3 Proofs

THEOREM 1. *The algorithms $(T, (U_1, U_2))$ of "Algorithm A" are an outsource-secure implementation of a pairing evaluation under the OMTUP assumption where the input (A, B) may be honest, secret, honest, protected, or adversarial, protected.*

PROOF. The correctness is obvious from the following equation:

$$\begin{aligned} o &= \alpha_1\alpha'_1\alpha'_2 e(P_1, P_2)^{x_5+x_6-x_2} \\ &= e(A, B)e(P_1, P_2)^{x_2}e(P_1, P_2)^{-x_5} \\ &\quad e(P_1, P_2)^{-x_6}e(P_1, P_2)^{x_5+x_6-x_2} \\ &= e(A, B) \end{aligned} \tag{10}$$

Next, we focus on the security aspect.

- $EVIEW_{real} \sim EVIEW_{ideal}$:

  For a round i, if the input (A, B) is protected or unprotected, the simulator S_1 behaves in the same way as in a real round. Else if (A, B) is an honest, secret input, the simulator S_1 behaves as follows. S_1 randomly selects points $(t_1P_1, t_2P_2, t_3P_1, t_4P_2)$ for U_1 to form random queries. S_1 also randomly selects points $(t_5P_1, t_6P_2, t_7P_2, t_8P_2, t_9P_1, t_{10}P_2)$ for U_2 to form random queries together with t_3P_1, and t_4P_2. More clearly,

  - $U_1(t_1P_1, t_2P_2) \rightarrow \alpha_1$;
  - $U_1(t_3P_1, t_4P_2) \rightarrow \alpha_2$;
  - $U_2(t_5P_1, t_6P_2) \rightarrow \alpha'_1$;
  - $U_2(t_7P_1, t_8P_2) \rightarrow \alpha'_2$;
  - $U_2(t_3P_1, t_4P_2) \rightarrow \alpha'_3$;
  - $U_2(t_9P_1, t_{10}P_2) \rightarrow \alpha'_4$.

  Next, according to the responses of U_1 and U_2, S_1 behaves as follows.

  - If $\alpha_2 \neq \alpha'_3$ or $e(t_9P_1, t_{10}P_2) \neq \alpha'_4$, S_1 produces $Y^i_p =$"Error", $Y^i_u = \emptyset$ and $rep^i = 1$.
  - If all responses are correct, S_1 sets $Y^i_p = \emptyset$, $Y^i_u = \emptyset$ and $rep^i = 0$.
  - Otherwise, S_1 selects a random value $o_r \in \mathbb{G}_T$ and sets $Y^i_p = o_r$, $Y^i_u = \emptyset$ and $rep^i = 1$.

  For each case, S_1 saves the appropriate states.

  The distributions of inputs in the real and ideal experiments are computationally indistinguishable for U_1 and U_2. In the ideal experiment, the inputs to U_1 or U_2 are chosen uniformly at random. In a real experiment, the inputs to U_1 or U_2 are independently re-randomized. Note that for U_1 or U_2, each input has a new random value that is different to the random values of other inputs.

  If U_1 and U_2 behave honestly in the round i, S_1 uses the Alg to give the correct output that is the same as the output of T^{U_1,U_2}. If U_1 and U_2 are dishonest, with a probability $1/2$, S_1 and T produce "Error". With a probability $1 - 1/2$, S_1 and T are cheated to produce a random output. Note that U_1 and U_2's responses are multiplied by a random value that is known only by T or S_1, and the party E knows nothing about the honest, secret input. Then we conclude that $EVIEW^i_{real} \sim EVIEW^i_{ideal}$, and then $EVIEW_{real} \sim EVIEW_{ideal}$ by a hybrid argument.

- $UVIEW_{real} \sim UVIEW_{ideal}$:

  For a round i, the simulator S_2 uses the same strategy as S_1 to produce random queries for U_1 and U_2, and saves its states and the states of U_1 and U_2. Note that E could not communicate directly with U_1 and U_2 after they are installed on a computing device. So E could not tell U_1 and U_2 that the simulator produces nothing as an output. And U_1, U_2 could not collaborate to test the random inputs since they are noninteractive. Then U_1 and U_2 could not distinguish random queries

from real queries for the same reason as they interact with S_1. Then we conclude that $UVIEW^i_{real} \sim UVIEW^i_{ideal}$, and then $UVIEW_{real} \sim UVIEW_{ideal}$ by a hybrid argument.

□

THEOREM 2. *The algorithms $(T,(U_1,U_2))$ of "Algorithm A" are a $(\mathcal{O}(\frac{1}{logq}), 1/2)$-outsource secure implementation of a pairing evaluation under the OMTUP assumption .*

PROOF. The computation of algorithm A includes a call to *RandA*, 4 point additions in $\mathbb{G}_1$ and $\mathbb{G}_2$, and 3 multiplications in $\mathbb{G}_T$. Since we use a dynamic table, the call to *RandA* is a table-lookup operation and the online cost could be omitted. As noted in [12], it takes roughly $\mathcal{O}(logq)$ multiplications to compute a bilinear pair. So the algorithms $(T,(U_1,U_2))$ are a $\mathcal{O}(1/logq)$-efficient secure outsourcing.

U_1 could not distinguish a test query from a real query. Note that we are under the OMTUP assumption, at least one response about the test query is assumed to be honest. So if U_1 is dishonest, it fails with a probability 1/2. U_2 could not distinguish the two test queries from the real queries. One test query is verified by the party T under the OMTUP assumption. The other is verified by the party T with a pre-computed value. So if U_2 is dishonest, it fails with a probability 1/2. □

4. A FLEXIBLE ALGORITHM

This section includes the construction of "Algorithm B" and its related security model and proofs.

4.1 Construction of Algorithm B

Algorithm B uses the same symbols as the algorithm A.

- *Init*: T calls *RandB* two times to produce values

$$(x_1P_1, x_1x_2^{-1}x_3P_1,\\ x_1^{-1}x_2P_2, x_1^{-1}x_4P_2\\ e(P_1,P_2)^{x_3+x_4-x_2})$$

and

$$(x'_1P_1, x'_1x'^{-1}_2x'_3P_1,\\ x'^{-1}_1x'_2P_2, x'^{-1}_1x'_4P_2\\ e(P_1,P_2)^{x'_3+x'_4-x'_2}).$$

It randomly selects a small integer $t \in \{1,\ldots,s\}$.

- *Computation*: T queries U_1 in a random order
  1. $U_1(A + x_1P_1, B + x_1^{-1}x_2P_2) \to \alpha_1$;
  2. $U_1(tA + x'_1x'^{-1}_2x'_3P_1, -x'^{-1}_1x'_2P_2) \to \alpha_2$;
  3. $U_1(-x'_1P_1, B + x'^{-1}_1x'_4P_2) \to \alpha_3$;

  Similarly, T queries U_2 in a random order
  1. $U_2(tA + x'_1P_1, B + x'^{-1}_1x'_2P_2) \to \alpha'_1$;
  2. $U_2(A + x_1x_2^{-1}x_3P_1, -x_1^{-1}x_2P_2) \to \alpha'_2$;
  3. $U_2(-x_1P_1, B + x_1^{-1}x_4P_2) \to \alpha'_3$;

- *Recover*: T computes

$$o = \alpha_1\alpha'_2\alpha'_3e(P_1,P_2)^{x_3+x_4-x_2}$$

and

$$o' = \alpha'_1\alpha_2\alpha_3e(P_1,P_2)^{x'_3+x'_4-x'_2}.$$

If $o^t = o'$ and $o \in \mathbb{G}_T$, T produces o as an output. Otherwise, it rejects and produces "Error".

REMARK 4.1. *According to a comment in [8], the membership test of an output is necessary. It is intended to ensure that a malicious program does not modify a response by multiplying an element of small orders. For some parameter sets [23, 1], the membership test does not need an exponentiation. A more general solution is to outsource the membership test operation. Since U_1 or U_2 knows the order q, we may use a secure public-exponent-secret-base outsourcing algorithm [34] to compute o^q and compare the result with $e(P_1,P_2)$ to determine whether $o \in \mathbb{G}_T$. Note that since q is the order of $\mathbb{G}_T$, the blind factors in [34] should not be selected from $\mathbb{G}_T$.*

We further propose the following wrapper protocol for membership test in the setting of elliptic curve pairings where $\mathbb{G}_T$ is a subgroup of $\mathbb{F}^*_{p^\lambda}$. Note p is a prime number and λ is the embedding degree that is the smallest integer such that $q|p^\lambda - 1$. We use a public-exponent-public-base exponentiation outsourcing algorithm [26] as a subroutine.

- *Off-line Phase: T randomly selects $u \in \mathbb{F}^*_{p^\lambda}$ and computes u^q.*
- *On-line Phase:*
  1. *T computes $v = uo$ and calls a public-exponent-public-base outsourcing algorithm to compute v^q.*
  2. *T checks whether $v^q = u^q$. If the equation holds, T believe that $o \in \mathbb{G}_T$.*

The simple protocol adds two online multiplications in $\mathbb{F}^*_{p^\lambda}$. The checkability is the same as that of the subroutine.

4.2 Security Model

Note that all definitions in Section 3.2 except the Definition 6 do not consider the number and trustability of programs, and they could be reused here. To emphasize the TUP assumption, we give the following definition.

Definition 7. If the party U denotes two noninteractive programs (U_1,U_2), and if U_1, U_2 are malicious, the algorithms (T,U) are implemented under an TUP assumption.

4.3 Proofs

THEOREM 3. *The algorithms $(T,(U_1,U_2))$ of "Algorithm B" are an outsource-secure implementation of a pairing evaluation under the TUP assumption where the input (a,u) may be honest, secret, honest, protected, or adversarial, protected.*

PROOF. The correctness is obvious from the following two equations:

$$\begin{aligned} o &= \alpha_1\alpha'_2\alpha'_3e(P_1,P_2)^{x_3+x_4-x_2}\\ &= e(A,B)e(P_1,P_2)^{x_2}e(P_1,P_2)^{x_3+x_4-x_2}\\ &\quad e(P_1,P_2)^{-x_3}e(P_1,P_2)^{-x_4}\\ &= e(A,B) \end{aligned} \tag{11}$$

$$\begin{aligned} o' &= \alpha'_1\alpha_2\alpha_3e(P_1,P_2)^{x'_1+x'_2-1}\\ &= e(A,B)^te(P_1,P_2)^{x'_2}e(P_1,P_2)^{x'_3+x'_4-x'_2}\\ &\quad e(P_1,P_2)^{-x'_3}e(P_1,P_2)^{-x'_4}\\ &= e(A,B)^t \end{aligned} \tag{12}$$

Next, we focus on the security aspect.

- $EVIEW_{real} \sim EVIEW_{ideal}$:

  For a round i, if the input (A, B) is protected or unprotected, the simulator S_1 behaves in the same way as in a real round. Else if (A, B) is an honest, secret input, the simulator S_1 behaves as follows. S_1 randomly selects points $(t_1P_1, t_2P_2, t_3P_1, t_4P_2, t_5P_1, t_6P_2)$ and $t \in \{1, \ldots, s\}$ for U_1 and U_2 to form random queries. More clearly,

  - $U_1(t_1P_1, t_2P_2) \rightarrow \alpha_1$;
  - $U_1(tt_3P_1, t_4P_2) \rightarrow \alpha_2$;
  - $U_1(tt_5P_1, t_6P_2) \rightarrow \alpha_3$;
  - $U_2(tt_1P_1, t_2P_2) \rightarrow \alpha'_1$;
  - $U_2(t_3P_1, t_4P_2) \rightarrow \alpha'_2$;
  - $U_2(t_5P_1, t_6P_2) \rightarrow \alpha'_3$;

  Next, according to the responses of U_1 and U_2, S_1 behaves as follows.

  - If $(\alpha_1\alpha'_2\alpha'_3)^t \neq \alpha'_1\alpha_2\alpha_3$, S_1 produces $Y_p^i =$"Error", $Y_u^i = \emptyset$ and $rep^i = 1$.
  - If all responses are correct, S_1 sets $Y_p^i = \emptyset$, $Y_u^i = \emptyset$ and $rep^i = 0$.
  - Otherwise, S_1 selects a random value $o_r \in \mathbb{G}_T$ and sets $Y_p^i = o_r$, $Y_u^i = \emptyset$ and $rep^i = 1$.

  For each case, S_1 saves the appropriate states.

  The distributions of inputs in the real and ideal experiments are computationally indistinguishable for U_1 and U_2. In the ideal experiment, the inputs to U_1 or U_2 are chosen uniformly at random. In a real experiment, the inputs to U_1 or U_2 are independently re-randomized. Note that for U_1 or U_2, each input has a new random value that is different to the random values of other inputs.

  If U_1 and U_2 behave honestly in the round i, S_1 uses the Alg to give the correct output that is the same as the output of T^{U_1,U_2}. If U_1 and U_2 are dishonest, with a probability about $(1-1/3s)^2$, S_1 and T produce "Error". With a probability about $1/3s(2-1/3s)$, S_1 and T are cheated to produce a random output. Note that in a real experiment, U_1 and U_2's responses are multiplied by a random value that is known only by T, and the party E knows nothing about the honest, secret input (A, B). In an ideal experiment, U_1 and U_2's responses are checked by the simulator. If their responses pass the check equation and are not correct, a random output is then produced, which means that U_1 or(and) U_2 has(have) guessed the value t and the order of inputs, and returns wrong results. Then we conclude that $EVIEW_{real}^i \sim EVIEW_{ideal}^i$, and then $EVIEW_{real} \sim EVIEW_{ideal}$ by a hybrid argument.

- $UVIEW_{real} \sim UVIEW_{ideal}$:

  For a round i, the simulator S_2 uses the same strategy as S_1 to produce random queries for U_1 and U_2, and saves its states and the states of U_1 and U_2. Note that E could not communicate directly with U_1 and U_2 after they are installed on a computing device. So E could not tell U_1 and U_2 that the simulator produces nothing as an output. And U_1, U_2 could not collaborate to obtain extra information about the inputs since they are noninteractive. Then U_1 and U_2 could not distinguish random inputs from real inputs. Then we conclude that $UVIEW_{real}^i \sim UVIEW_{ideal}^i$, and then $UVIEW_{real} \sim UVIEW_{ideal}$ by a hybrid argument.

□

THEOREM 4. *The algorithms $(T, (U_1, U_2))$ of "Algorithm B" are a $(\mathcal{O}(\frac{logs}{logq}), (1-1/3s)^2)$-outsource secure implementation of a pairing evaluation.*

PROOF. Algorithm B includes two calls to the $RandB$ algorithm, and $\mathcal{O}(logt)$ point additions and $\mathcal{O}(logt)$ modular multiplications. We use a dynamic table and the calls to $RandB$ are table-lookup operations. We then claim that the online computation is about $\mathcal{O}(logs)$ since $t \in \{1, \ldots, s\}$. As we noted in Theorem 2, it takes roughly $\mathcal{O}(logq)$ multiplications to compute a bilinear pair. So the algorithms $(T, (U_1, U_2))$ are a $\mathcal{O}(logs/logq)$-efficient secure outsourcing.

We assume two malicious U_1 and U_2 without interactions. The checking equation $o^t = o'$ could be written as

$$\begin{aligned} & (e(P_1, P_2)^{x_3+x_4-x_2})^t \alpha_1^t \alpha_2^{-1} \alpha_3^{-1} \\ = \; & e(P_1, P_2)^{x'_3+x'_4-x'_2} \alpha'_1 \alpha'^{-t}_2 \alpha'^{-t}_3. \end{aligned} \tag{13}$$

In the right side of the equation (13), if U_2 wants to change α'_2 or α'_3 and to keep the equation holding, it should know the value t. In the left side, if U_1 wants to change the product of $\alpha_2\alpha_3$ and to keep the equation holding, it should know the value t. If U_2 changes only α'_1 and U_1 changes only the product of $\alpha_2\alpha_3$ and they keep the equation holding, then the product of $\alpha'_1\alpha_2\alpha_3$ keeps unchanged and the output is correct.

Now suppose U_i, $i \in_R \{1, 2\}$ guesses a correct value t. Considering U_1, it now has to identify the query to produce α_1 out of the three queries. Since the inputs to U_1 are in a random order, the probability is $1/3$ for U_1 to identify the query. Similarly, U_2 has to identify α'_1 with a probability $1/3$.

In summary, the chance for U_1 or U_2 to change their outputs and to keep the verification equation holding is to guess the value t and the inputting order correctly. The probability of successful checking is then $(1-1/3s)^2$. □

5. COMPARISON

We show two tables to compare the efficiency and security properties between our algorithms and two up-to-date algorithms [8, 12]. The "Alg. [8]" refers to their main protocol for public points outsourcing. They propose a general conversion for secure bilinear pairing delegation. That conversion adds 2 scalar multiplications and 1 modular exponentiation, which leads to a more inefficient implementation. For "Algorithm B", we set $s = 4$ to give a concrete example about computations and computations. Note again the value s is configurable. The symbols SM, ME and PA are defined in Section 2.6 denoting a scalar multiplication, a modular exponentiation and a point addition. The symbol MT denotes a membership test operation, and MM a modular multiplication. From Table 2, it is obvious that our "Algorithm A" has the best efficiency.

Table 3 shows the properties of different algorithms. It is obvious the algorithm in [8] is under a convenient OUP assumption and has the best checkability. However, the works

Table 2: Efficiency Comparison of Up-to-date Algorithms

	SM	ME	MT	PA	MM
Alg. [8]	2	1	1	2	1
Alg. [12]	0	0	0	5	4
Alg. A	0	0	0	4	3
Alg. B	0	0	1	$\mathcal{O}(logs)$	$\mathcal{O}(logs)$
Alg. B ($s = 4$)	0	0	1	10	8

Table 3: Properties Comparison of Up-to-date Algorithms

	Assumptions	Secrecy	Checkability
Alg. [8]	OUP	No	1
Alg. [12]	OMTUP	YES	1/2
Alg. A	OMTUP	YES	1/2
Alg. B	TUP	YES	$(1 - 1/3s)^2$
Alg. B ($s = 4$)	TUP	YES	0.84

in [25, 24] shows that their algorithm is meaningful for only some parameters. Comparatively, our "Algorithm B" shows an alternative approach to balance the security properties and efficiency. The algorithm needs no scalar multiplications or modular exponentiations as the algorithm in [12]. And the client needs not to check the vendor of a program to satisfy the OMTUP assumption. And a client could set s larger to get a better checkability at the cost of more computations. So we claim the algorithm B as a more flexible algorithm. If $s = 4$, the probability to check a result is about 0.84. Note that the "Algorithm B" is under the TUP assumption. If the algorithm [12] is under the same TUP assumption, their checkability is about 0.25.

6. CONCLUSION

This paper showed two algorithms for bilinear pairing outsourcing. One is more efficient than the state-of-the-art algorithms. The other is more flexible to be deployed in a really two-malicious-program environment, and enables an approach to balance the computation cost and the checkability. It also showed two pre-computation algorithms to compute off-line random vectors. Further, we are implementing our algorithms for low power devices to study the computation, memory and communication costs in practice with real world examples.

7. ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under No. 61379154 and U1135001, Huawei Grant YB2013120027, and US National Science Foundation under grant CNS-1262277.

8. REFERENCES

[1] Paulo S L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography*, pages 319–331, 2006.

[2] Chevallier-Mames Benoit, Coron Jean-Sebastien, McCullagh Noel, Naccache David, and Scott Michael. Secure delegation of elliptic-curve pairing. Cryptology ePrint Archive, Report 2005/150, 2005. http://eprint.iacr.org/.

[3] Kang Bo Gyeong, Lee Moon Sung, and Park Je Hong. Efficient delegation of pairing computation. Cryptology ePrint Archive, Report 2005/259, 2005. http://eprint.iacr.org/.

[4] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Berlin Heidelberg, 2001.

[5] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[6] Victor Boyko, Marcus Peinado, and Ramarathnam Venkatesan. Speeding up discrete log and factoring based schemes via precomputations. In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT'98*, pages 221–235, 1998.

[7] SÍębastien Canard, Iwen Coisel, Julien Devigne, ClÍęcilia Gallais, Thomas Peters, and Olivier Sanders. Toward generic method for server-aided cryptography. In Sihan Qing, Jianying Zhou, and Dongmei Liu, editors, *Information and Communications Security*, volume 8233 of *Lecture Notes in Computer Science*, pages 373–392. Springer International Publishing, 2013.

[8] SÍębastien Canard, Julien Devigne, and Olivier Sanders. Delegating a pairing can be both secure and efficient. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security*, pages 549–565, 2014.

[9] David Chaum and TorbenPryds Pedersen. Wallet databases with observers. In ErnestF Brickell, editor, *Advances in Cryptology - CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer Berlin Heidelberg, 1993.

[10] Xiaofeng Chen, Jin Li, Jianfeng Ma, Qiang Tang, and Wenjing Lou. New algorithms for secure outsourcing of modular exponentiations. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *Computer Security ÍC ESORICS 2012*, pages 541–556, 2012.

[11] Xiaofeng Chen, Jin Li, Jianfeng Ma, Qiang Tang, and Wenjing Lou. New algorithms for secure outsourcing of modular exponentiations. *Ieee Transactions on Parallel And Distributed Systems*, 25(9):2386–2396, 2014.

[12] Xiaofeng Chen, Willy Susilo, Jin Li, Duncan S. Wong, Jianfeng Ma, Shaohua Tang, and Qiang Tang. Efficient algorithms for secure outsourcing of bilinear pairings. *Theoretical Computer Science*, (In Press), 2014.

[13] Benoit Chevallier-Mames, Jean-SÍębastien Coron, Noel McCullagh, David Naccache, and Michael Scott. Secure delegation of elliptic-curve pairing. In Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny, editors, *Smart Card Research and Advanced Application*, pages 24–35, 2010.

[14] JaeCha Choon and Jung Hee Cheon. An identity-based signature from gap diffie-hellman groups. In YvoG Desmedt, editor, *Public Key Cryptography - PKC 2003*, volume 2567 of *Lecture*

Notes in Computer Science, pages 18–30. Springer Berlin Heidelberg, 2002.

[15] Sherman S. M. Chow, Man Ho Au, and Willy Susilo. Server-aided signatures verification secure against collusion attack. *Information Security Technical Report*, 17(3):46–57, 2013.

[16] Marc Girault and David Lefranc. Server-aided verification: Theory and practice. In Bimal Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, pages 605–623, 2005.

[17] Susan Hohenberger and Anna Lysyanskaya. How to securely outsource cryptographic computations. In Joe Kilian, editor, *Theory of Cryptography*, pages 264–282, 2005.

[18] ChaeHoon Lim and PilJoong Lee. Server(prover/signer)-aided verification of identity proofs and signatures. In LouisC Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology - EUROCRYPT'95*, volume 921 of *Lecture Notes in Computer Science*, pages 64–78. Springer Berlin Heidelberg, 1995.

[19] Tsutomu Matsumoto, Koki Kato, and Hideki Imai. Speeding up secret computations with insecure auxiliary devices. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTOá88*, volume 403 of *Lecture Notes in Computer Science*, pages 497–506. Springer New York, 1990.

[20] PhongQ Nguyen, IgorE Shparlinski, and Jacques Stern. Distribution of modular sums and the security of the server aided exponentiation. In Kwok-Yan Lam, Igor Shparlinski, Huaxiong Wang, and Chaoping Xing, editors, *Cryptography and Computational Number Theory*, volume 20 of *Progress in Computer Science and Applied Logic*, pages 331–342. Birkhauser Basel, 2001.

[21] Tsang Patrick P., Chow Sherman S. M., and Smith Sean W. Batch pairing delegation. In *Proceedings of the Security 2nd international conference on Advances in information and computer security*, pages 74–90, Nara, Japan, 2007.

[22] Kui Ren, Cong Wang, and Qian Wang. Security challenges for the public cloud. *Internet Computing, IEEE*, 6(1):69–73, 2012.

[23] Michael Scott. Unbalancing pairing-based key exchange protocols. Cryptology ePrint Archive, Report 2013/688, 2013. http://eprint.iacr.org/.

[24] Michael Scott, Neil Costigan, and Wesam Abdulwahab. Implementing cryptographic pairings on smartcards. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 134–147, 2006.

[25] AnaHelena Slćnchez and Francisco RodrĺŁguez-HenrĺŁquez. Neon implementation of an attribute-based encryption scheme. In Michael Jacobson, Michael Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, pages 322–338, 2013.

[26] Marten Van Dijk, Dwaine Clarke, Blaise Gassend, G. Edward Suh, and Srinivas Devadas. Speeding up exponentiation using an untrusted computational resource. *Designs, Codes and Cryptography*, 39(2):253–273, 2006.

[27] Cong Wang, Kui Ren, and Jia Wang. Secure and practical outsourcing of linear programming in cloud computing. In *INFOCOM, 2011 Proceedings IEEE*, pages 820–828, 2011.

[28] Yujue Wang, Qianhong Wu, DuncanS Wong, Bo Qin, ShermanS M. Chow, Zhen Liu, and Xiao Tan. Securely outsourcing exponentiations with single untrusted program for cloud storage. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, pages 326–343, 2014.

[29] Zhiwei Wang, Licheng Wang, Yixian Yang, and Zhengming Hu. Comment on wu et al.'s server-aided verification signature schemes. *International Journal of Network Security*, 10(3):238–240, 2010.

[30] Wei Wu, Yi Mu, Willy Susilo, and Xinyi Huang. Server-aided verification signatures: Definitions and new constructions. In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *Provable Security*, pages 141–155, 2008.

[31] Wei Wu, Yi Mu, Willy Susilo, and Xinyi Huang. Provably secure server-aided verification signatures. *Computers & Mathematics with Applications*, 61(7):1705–1723, 2011.

[32] Zhen Xu, Cong Wang, Qian Wang, Kui Ren, and Lingyu Wang. Proof-carrying cloud computation: The case of convex optimization. In *INFOCOM, 2013 Proceedings IEEE*, pages 610–614, 2013.

[33] Andrew C. Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on*, pages 160–164, 1982.

[34] Fangguo Zhang, Xu Ma, and Shengli Liu. Efficient computation outsourcing for inverting a class of homomorphic functions. *Information Sciences*, 286(1):19–28, 2014.