New Efficient and Flexible Algorithms for Secure Outsourcing of Bilinear Pairings

Xi-Jun Lin [*], Haipeng Qu [†]and Xiaoshuai Zhang [‡]

January 28, 2016

Abstract: Outsourcing paradigm has become a hot research topic in the cryptography community, where computation workloads can be outsourced to cloud servers by the resource-constrained devices, such as RFID tags. The computation of bilinear pairings is the most expensive operation in pairing-based cryptographic primitives. In this paper, we present two new algorithms for secure outsourcing the computation of bilinear pairings. One is secure in the OMTUP model. The other, which provides flexible checkability, is in the TUP model. Compared with the state-of-the-art algorithms, our proposal is more efficient.

Key words: secure outsourcing algorithm; bilinear pairings; one-malicious model

1 Introduction

Cloud computing is a new computing paradigm which enables IT capacities and resources to be provided as services over the Internet. In recent years, outsourcing paradigm has become a hot research topic in the cryptography community due to the advancement of cloud computing, where computation workloads can be outsourced to cloud servers by the resource-constrained devices, such as RFID tags. As a result, some new challenges [32, 36] have to be considered, which are shown as follows.

- Security: First of all, the cloud servers may be untrusted, while some sensitive information, which should not be leaked to the servers, may be contained in the outsourced data. Hence, as for the secrecy, it is required that the servers should learn nothing useful about what it is actually computing.
- Checkability: Some invalid results may be returned due to some cloud servers misbehaves. Therefore, the outsourcers should have the ability to verify any failures. It is clear that the verification must be far more efficient than the outsourced computing work itself.

[*]X.J.Lin is with the Department of Computer Science and Technology, Ocean University of China. Qingdao 266100, P.R.China. email: linxj77@163.com

[†]H.Qu is with the Department of Computer Science and Technology, Ocean University of China. Qingdao 266100, P.R.China.

[‡]X.Zhang is with the Department of Computer Science and Technology, Ocean University of China. Qingdao 266100, P.R.China.

- Security Models: The number and trustability of cloud servers is crucial to study the security of secure outsourcing algorithms. One untrusted program (OUP) model allows only one server, which may be malicious, to implement an algorithm. One-malicious (OMTUP) model (i.e., one-malicious version of two untrusted program) allows two servers to implement an algorithm, and only one of them could be malicious. Two untrusted program (TUP) model allows two servers to implement an algorithm and both of them could be malicious.

On the other hand, the bilinear pairings have become a hot technique in cryptography. However, the computation of bilinear pairings is still expensive for some resource-constrained devices although some efforts have been made to improve its efficiency. Hence, how to securely outsource such computation has become a hot research topic in cryptography community.

1.1 Related Work

The impossibility of secure outsourcing an exponential computation while locally doing only polynomial time work was proved in [2]. In the computer theory community, the problem of secure outsourcing expensive computations, such as matrix multiplications and quadrature [3], sequence comparisons [4, 10], linear algebra computations [1, 7] and linear programming computations [37], receives considerable attentions.

On the other hand, the problem of secure outsourcing expensive computations receives more and more attentions in the cryptographic community. The first idea with respect to this problem was introduced in the so-called "wallets with observers" by Chaumand Pedersen [17]. The security model for outsourcing cryptographic computations was first presented by Hohenberger and Lysyanskaya [28] who proposed the first outsource-secure algorithm for modular exponentiations based on server-aided computation [5, 23, 29, 38] and precomputation [13, 21, 30, 33].

The notion of ringers to verify computation completion was introduced in [26], which was followed by the researches [9, 15, 16, 34]. The notion of verifiable outsourcing computation for arbitrary functions was first formalized by Gennaro et al. [22], which was followed by plenty of researches [11, 12, 24, 25, 35]. In [22], a protocol which allows the outsourcer to check the outputs with a non-interactive, computationally sound proof was proposed. Following the idea in [22], the first practical verifiable computation scheme for high degree polynomial functions was proposed in [6]. And new approaches were presented by Green et al. [27] for constructing secure outsourcing attribute-based encryption (ABE), which was followed by Parno et al. [31] who proposed a multi-function verifiable computation scheme.

The first algorithm for secure delegation of elliptic-curve pairings in the OUP model was presented in [19], while the disadvantage is that some other expensive operations have to be carried out by the outsourcer although any server misbehave can be detected with probability 1.

Recently, Chen et al. [18] presented an outsource-secure algorithm of bilinear pairings in the OMTUP model. Compared with the algorithm in [19], a distinguishing property of their proposed algorithm is that the resource-constrained outsourcer never needs to perform any expensive operations. However, some expensive operations, such as scalar multiplications and bilinear pairings, have to be performed in the pre-computation.

Tian et al. [35] proposed two algorithms, called Algorithm A and Algorithm B respectively, for secure bilinear pairing outsourcing. Algorithm A, which is also in the OMTUP model, improves the efficiency of pre-computation compared with that in [18]. Algorithm B, which is in the TUP model, provides a client a choice to improve the checkability.

Note that an algorithm may require more computations in the OUP model compared with the OMTUP and TUP models. As for some practical scenarios, the TUP model may be more well-suited than the OMTUP model. This is because many cloud services may be based on the same platform provided by a vendor. Also note that an algorithm secure in the OMTUP model may be insecure in the TUP model.

1.2 Our Contribution

In this paper, we propose two secure outsourcing algorithm for bilinear pairings, called **Pair** and **FPair** respectively. Compared with the state-of-the-art algorithms [18, 35], our proposals have the following advantages:

The algorithm **Pair** is in the OMTUP model. Compared with the state-of-the-art algorithms (i.e., algorithm in [18] and Algorithm A in [35], which are both in the OMTUP model), **Pair** is more efficient in both the computation of T and pre-computation. On the other hand, **FPair**, which provides flexible checkability, is in the TUP model, where the pre-computation is more efficient than the state-of-the-art algorithm, i.e., Algorithm B [35] (which is also in the TUP model).

1.3 Organization

The rest of the paper is organized as follows: In Section 2, some security definitions for outsourcing computation are recalled. The new and efficient secure outsourcing algorithms **Pair** and **FPair** for bilinear pairings are presented in Section 3 and Section 4 respectively, which is followed by the last section to conclude our work.

2 Preliminaries

2.1 Bilinear pairings

Let G_1 and G_2 be two cyclic additive groups of a large prime order q generated by P_1 and P_2, respectively. Let G_T be a cyclic multiplicative group of the same order q. A bilinear pairing is a map $e : G_1 \times G_2 \to G_T$ with the following properties:

1. Bilinear: $e(aR, bQ) = e(R, Q)^{ab}$ for all $R \in G_1, Q \in G_2$, and $a, b \in \mathbb{Z}_q^*$.
2. Non-degenerate: There exist $R \in G_1$ and $Q \in G_2$ such that $e(R, Q) \neq 1$.
3. Computable: There is an efficient algorithm to compute $e(R, Q)$ for all $R, Q \in G_1$.

2.2 Definition of Outsource-Security

Let Alg be a cryptographic algorithm. Roughly speaking, some work was securely outsourced to the cloud server U by an honest device T, and (T, U) is an outsource-secure implementation of Alg if 1) T and U implement Alg, i.e., $Alg = T^U$ and 2) the adversary U' can learn nothing about the input/output of $T^{U'}$ if T is allowed to make oracle access to U'. Note that U' may record all of its computation and may be malicious. The formal definitions [28] are recalled 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'. 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 $\mathcal{A}$.*

As for the security, E should learn nothing useful about the secret inputs/outputs of T^U even if the malicious software U' is written by E, which is ensured by the following definition for outsource-security proposed in [28]:

Definition 2 (Outsource-Security) *Let Alg be an algorithm with outsource-I/O. A pair of algorithms (T, U) are an outsource-secure implementation 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. $EVIEW_{real} \sim EVIEW_{ideal}$:*
     - *The view that the adversarial environment E obtains by participating in the following real process:*

$$\begin{aligned} EVIEW_{real}^{i} &= \{(istate^i, x_{hs}^i, x_{hp}^i, x_{hu}^i) \\ &\leftarrow I(1^k, istate^{i-1}); \\ &\quad (estate^i, j^i, x_{ap}^i, x_{au}^i, stop^i) \\ &\leftarrow E(1^k, EVIEW_{real}^{i-1}, x_{hp}^i, x_{hu}^i); \\ &\quad (tstate^i, ustate^i, y_s^i, y_p^i, y_u^i) \\ &\leftarrow T^{U'(ustate^{i-1})} \\ &\quad (tstate^{i-1}, x_{hs}^{j^i}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^i, \\ &\quad x_{au}^i) : (estate^i, y_p^i, y_u^i)\} \end{aligned}$$

$EVIEW_{real} \sim EVIEW_{real}^i$ if $stop^i = TRUE$.
The real process proceeds in rounds. In the i-th round, the honest (secret, protected, and unprotected) inputs $(x_{hs}^i, x_{hp}^i, x_{hu}^i)$ 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

(a) *the value of its* $estate^i$ *variable as a way of remembering what it did next time it is invoked;*

(b) *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);*

(c) *the adversarial, protected input* x^i_{ap}*;*

(d) *the adversarial, unprotected input* x^i_{au}*;*

(e) *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* $(tstate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$*, 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^i_s*, protected* y^i_p *and unprotected* y^i_u *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^i_p *and* y^i_u*. The overall view of* E *in the real process is just its view in the last round (i.e.,* i *for which* $stop^i = TRUE$*.).*

– *The ideal process:*

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

$EVIEW_{ideal} = EVIEW^i_{ideal}$ *if* $stop^i = TRUE$*:*
The ideal process also proceeds in rounds. In the ideal process, there is a 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 the indicator variable* $replace^i$ *is a bit for determining whether* y^i_p *will be replaced with* Y^i_p*. Hence, it is allowed to query oracle* U'*; moreover,* U' *saves its state as in the real experiment.*

- *Pair Two.* $UVIEW_{real} \sim UVIEW_{ideal}$*:*
  – *The view that the untrusted software* U' *obtains by participating in the real process described in* Pair One. $UVIEW_{real} = ustate^i$ *if* $stop^i = TRUE$*.*

– *The ideal process:*

$$\begin{array}{rcl} UVIEW^i_{ideal} & = & \{(istate^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \\ & \leftarrow & I(1^k, istate^{i-1}); \\ & & (estate^i, j^i, x^i_{ap}, x^i_{au}, stop^i) \\ & \leftarrow & E(1^k, estate^{i-1}, x^i_{hp}, x^i_{hu}, \\ & & y^i_p, y^i_u); \\ & & (astate^i, y^i_s, y^i_p, y^i_u) \\ & \leftarrow & Alg(astate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, \\ & & x^{j^i}_{hu}, x^i_{ap}, x^i_{au}); \\ & & (sstate^i, ustate^i) \leftarrow \\ & & S_2^{U'(ustate^{i-1})}(sstate^{i-1}, \\ & & x^{j^i}_{hu}, x^i_{au}) : (ustate^i)\} \end{array}$$

$UVIEW_{ideal} = UVIEW^i_{ideal}$ *if* $stop^i = TRUE$:
In the ideal process, we have a stateful simulator S_2 *who, equipped with only the unprotected inputs* (x^i_{hu}, x^i_{au}), *queries* U'. *As before,* U' *may maintain state.*

Definition 3 (α-Efficient, Secure Outsourcing) *The algorithms* (T, U) *are 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) *The algorithms* (T, U) *are a* 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 probability no less than* β.

Definition 5 ((α, β)-Outsource-Security) *The algorithms* (T, U) *are an* (α, β)*-outsource-secure implementation of Alg if they are both* α*-efficient and* β*-checkable.*

According to the number and trustability of U, the OMTUP and TUP models are defined as follows [35]:

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

Definition 7 (TUP Model) *If* U *denotes two noninteractive programs* (U_1, U_2), *and if* U_1, U_2 *are malicious, the algorithms* (T, U) *are implemented in an TUP model.*

2.3 Notations

Let PA denote a point addition in G_1 (or G_2), SM denote a scalar multiplication in G_1 (or G_2), M denote a modular multiplication in G_T or $(\mathbb{Z}_q^*)$, Exp denote a modular exponentiation in G_T (or $\mathbb{Z}_q^*$), $MInv$ denote a modular inverse in $\mathbb{Z}_q^*$, and P denote a computation of the bilinear pairings.

3 New Efficient Outsourcing Algorithm of Bilinear Pairings

3.1 Pre-computation

In [18, 35], a subroutine *Rand* is used to speed up the computations. Similarly, we also use such a subroutine. The output for each invocation of *Rand* is a random, independent tuple $(V_1, V_2, vV_1, vV_2, V_3, V_4, e(V_3, V_4), e(-V_1, V_2))$, where $V_1, V_3 \in G_1$, $V_2, V_4 \in G_2$ and $v \in \mathbb{Z}_q^*$. Let $\chi = e(V_3, V_4)$ and $\lambda = e(-V_1, V_2)$.

A naive approach to implement this functionality is for a trusted entity to compute a table in advance and then load it into the memory of T [18, 35]. For each invocation of *Rand*, T just retrieves a new tuple by using the so-called table-lookup method. An efficient method was proposed in [35] to create such a table by pre-processing a static table with the EBPV generator [30] which is a well-known preprocessing algorithm. [1] The details of the method is referred to [35].

Similarly, we also use this method to create the table. At first, we generate $e(P_1, P_2)$, and then for each tuple perform the following steps:

1. Following the same procedure in [35], create four pairs (v_1, V_1), (v_2, V_2), (v_3, V_3), (v_4, V_4), where $V_1 = v_1P_1, V_2 = v_2P_2, V_3 = v_3P_1$ and $V_4 = v_4P_2$; (It is noted in [30] that the computation cost of a pair is about $k+h-3$ point additions, where k is about twenties and h is less than 10. It is obvious that this step requires less computations compared with the computation of bilinear pairings.)

2. Pick a random number $v \in \mathbb{Z}_p^*$, and compute vV_1 and vV_2;

3. Compute $v' = v_3v_4 \pmod q$ and $v'' = -v_1v_2 \pmod q$;

4. Compute $\chi = e(P_1, P_2)^{v'}$ and $\lambda = e(P_1, P_2)^{v''}$. It is clear that $\chi = e(V_3, V_4)$ and $\lambda = e(-V_1, V_2)$.

5. Store the tuple $(V_1, V_2, vV_1, vV_2, V_3, V_4, \chi, \lambda)$ into the table.

3.2 The Proposed Algorithm

Here, we present a new and more efficient outsource-secure algorithm **Pair** for bilinear pairings in the OMTUP model. Similar to [18, 35], T outsources its computations to U_1 and U_2 by invoking *Rand*. The security requirement is that the adversary $\mathcal{A}$ cannot know any useful information about the inputs/outputs of **Pair**. Let $U_i(\Lambda, \Pi) \rightarrow e(\Lambda, \Pi)$ denote that U_i takes (Λ, Π) and outputs $e(\Lambda, \Pi)$, where $i = 1, 2$.

The input of **Pair** is two random points $A \in G_1, B \in G_2$, 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.

Our main trick is to logically split A and B into random looking pieces. Let $\alpha_1 =$

[1]Since the EBPV generator has no limitations on the cyclic groups, it could be trivially applied to the setting of bilinear pairings [18, 35]. It is conjectured that with a sufficiently large subset of truly random pairs the output distribution of the EBPV generator is statistically close to the uniform distribution.

$e(A-V_1, B-V_2)$, $\alpha_2 = e(V_1, B+vV_2)$ and $\alpha_3 = e(A-vV_1, V_2)$. Note that

$$\begin{aligned}\alpha_1 &= e(A,B)e(A,-V_2)e(-V_1,B)e(V_1,V_2)\\ \alpha_2 &= e(V_1,B)e(vV_1,V_2)\\ \alpha_3 &= e(A,V_2)e(-vV_1,V_2)\end{aligned}$$

Thus, $e(A,B) = \lambda\alpha_1\alpha_2\alpha_3$.

Hence, **Pair** consists of the following steps for outsourcing the computation of bilinear pairings:

1. Run *Rand* to obtain a random tuple $(V_1, V_2, vV_1, vV_2, V_3, V_4, \chi, \lambda)$;
2. Pick $X \in G_1$ and $Y \in G_2$ randomly;
3. Query U_1 in random order as

   $U_1(A-V_1, B-V_2) \to \alpha_1$;

   $U_1(X,Y) \to e(X,Y)$.

   Similarly, query U_2 in random order as

   $U_2(V_1, B+vV_2) \to \alpha_2$;

   $U_2(A-vV_1, V_2) \to \alpha_3$;

   $U_2(V_3, V_4) \to e(V_3, V_4)$;

   $U_2(X,Y) \to e(X,Y)$.
4. Finally, check whether or not both U_1 and U_2 produce the correct outputs, i.e., $U_1(X,Y) = U_2(X,Y)$ and $U_2(V_3,V_4) = \chi$. If it is the case, output $\lambda\alpha_1\alpha_2\alpha_3$; otherwise, "error".

3.3 Security Proof

Theorem 1 *In the OMTUP 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 correctness property is straight-forward, and we only prove the security. The proof is very similar to [18, 28]. Let $\mathcal{A} = (E, U_1', U_2')$ be a probabilistic polynomial-time adversary that interacts with a probabilistic polynomial-time algorithm T in the one-malicious model.

Firstly, we prove *Pair One* $EVIEW_{real} \sim EVIEW_{ideal}$ (The external adversary, E, learns nothing.):

If the input (A,B) is anything other than honest, secret, then the simulator S_1 trivially behaves the same way as in the real execution.

If (A,B) is an honest, secret input, S_1 performs the following steps: In the i-th round, S_1 ignores the input and instead makes three random queries $(P_j \in G_1, Q_j \in G_2)$ to U_1' and U_2'. S_1 randomly checks two outputs from each software (i.e., $e(P_j, Q_j)$). If there is an error, S_1 outputs $Y_p^i =$ "*error*", $Y_u^i = \phi$, $replace^i = 1$ and all states are saved. If there is no error, S_1 verifies the remaining outputs. If all checks pass, S_1 outputs $Y_p^i = \phi$, $Y_u^i = \phi$,

$replace^i = 0$; otherwise, S_1 picks r randomly and outputs $Y_p^i = r$, $Y_u^i = \phi$, $replace^i = 1$. In either case, the appropriate states is saved by S_1.

Note that the input distributions to (U_1', U_2') in the ideal and real experiments are computationally indistinguishable. In the ideal experiment, the inputs are chosen randomly. In the real experiment, all numbers in the queries made by T is computationally indistinguishable from random since they are independently re-randomized.

If (U_1', U_2') is honest in the i-th round, $EVIEW_{real}^i \sim EVIEW_{ideal}^i$ (since $T^{(U_1', U_2')}$ executes **Pair** in the real experiment perfectly and S_1 simulates with the same outputs in the ideal experiment). If one of (U_1', U_2') behaves dishonestly in the i-th round and it is detected by T and S_1 (with probability $\frac{1}{2}$), an output of "error" will be given; otherwise, the output of **Pair** will be successfully corrupted (with probability $\frac{1}{2}$). In the real experiment, the outputs generated by (U_1', U_2') are multiplied together along with a random value, thus a corrupted output of **Pair** looks random to E. In the ideal experiment, S_1 also simulates with a random value $r \in \mathbb{Z}_p^*$. Thus, $EVIEW_{real}^i \sim EVIEW_{ideal}^i$ even when one of (U_1', U_2') is dishonest. Hence, we can conclude that $EVIEW_{real} \sim EVIEW_{ideal}$.

Secondly, we prove *Pair Two* $UVIEW_{real} \sim UVIEW_{ideal}$ (The untrusted software, (U_1', U_2'), learns nothing.):

The simulator S_2 always performs as follows: In the i-th round, S_2 ignores the input and instead makes three random queries $(P_j \in G_1, Q_j \in G_2)$ 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 since the output in the ideal experiment is never corrupted. However, this information cannot be communicated to (U_1', U_2') by E since T always re-randomizes its inputs to (U_1', U_2') in the i-th round of the real experiment. In the ideal experiment, S_2 always makes random, independent queries for (U_1', U_2'). Thus, for each round i we have $UVIEW_{real}^i \sim UVIEW_{ideal}^i$. Hence, we can conclude that $UVIEW_{real} \sim UVIEW_{ideal}$. □

Theorem 2 *In the OMTUP model, the algorithms* $(T, (U_1, U_2))$ *are an* $(O(\frac{1}{logq}), \frac{1}{2})$*-outsource-secure implementation of* **Pair**.

Proof. The proposed algorithm **Pair** makes only 1 invocation of *Rand*, 4 point additions in G_1 (or G_2), and 3 modular multiplications in G_T in order to compute $e(A, B)$. Also, the computation for *Rand* is negligible since we use the table-lookup method. Moreover, it takes **Pair** roughly $O(logq)$ multiplications in resulting finite filed to compute the bilinear pairings. Hence, the algorithms $(T, (U_1, U_2))$ are an $O(\frac{1}{logq})$-efficient implementation of **Pair**.

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

3.4 Efficiency

Here, we compare our proposal with the state-of-the-art algorithms [18, 35]. The comparison of the efficiency is presented in Table 1. The comparison of pre-computation is shown in Table 2. Compared with the algorithm in [18], our proposal is obviously more efficient. Compared with Algorithm B in [35], our proposal is more efficient in pre-computation phase. On the other hand, shorter tuple is required in our proposal. Thus, more storage could be saved, which may be crucial to the resource-constrained devices.

Table 1: Efficiency Comparison

	Algorithm [18]	Algorithm A [35]	Ours (**Pair**)
T	5PA+ 4M	4PA+3M	4PA + 3M
U	8P	6P	6P
Tuple-size	18	10	8
Checkability	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$

Table 2: Comparison of Pre-computations

	Algorithm [18]	Algorithm A [35]	Ours
SM	9	3	2
Exp	-	2	2
M	-	5	2
MInv	-	2	-
PA	-	$5(k+h-3)$	$4(k+h-3)$
P	3	-	-

4 New Flexible Outsourcing Algorithm of Bilinear Pairings

4.1 Pre-computation

Similarly, a subroutine *Rand* is also used here to speed up the computations by using the table-lookup method. The output for each invocation of *Rand* is a random, independent tuple $(V_1, V_2, vV_1, vV_2, e(-V_1, V_2))$, where $V_1 \in G_1$, $V_2 \in G_2$ and $v \in \mathbb{Z}_q^*$. Let $\lambda = e(-V_1, V_2)$.

Also, the method for creating the table is very similar to that in Section 3.1. At first, we generate $e(P_1, P_2)$, and then for each tuple perform the following steps:

1. Following the same procedure in [35], create two pairs $(v_1, V_1), (v_2, V_2)$, where $V_1 = v_1 P_1$ and $V_2 = v_2 P_2$;
2. Pick a random number $v \in \mathbb{Z}_p^*$, and compute vV_1 and vV_2;
3. Compute $v' = -v_1 v_2 \pmod q$;
4. Compute $\lambda = e(P_1, P_2)^{v'}$. It is clear that $\lambda = e(-V_1, V_2)$.
5. Output $(V_1, V_2, vV_1, vV_2, \lambda)$.

4.2 The Proposed Algorithm

Here, we present a new flexible outsource-secure algorithm **FPair** for bilinear pairings. Similarly, the input of **FPair** is two random points $A \in G_1, B \in G_2$, and the output of **FPair** 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.

Our main trick is to logically split A and B into random looking pieces with a random small integer $t \in \{1, \cdots, s\}$. Let $\alpha_1 = e(A - V_1, B - V_2)$, $\alpha_2 = e(V_1, B + vV_2)$ and $\alpha_3 =$

$e(A - vV_1, V_2)$. Note that

$$\begin{array}{rcl} \alpha_1 & = & e(A,B)e(A,-V_2)e(-V_1,B)e(V_1,V_2) \\ \alpha_2 & = & e(V_1,B)e(vV_1,V_2) \\ \alpha_3 & = & e(A,V_2)e(-vV_1,V_2) \end{array}$$

Thus, $o = e(A,B) = \lambda\alpha_1\alpha_2\alpha_3$.

On the other hand, let $\alpha'_1 = e(tA - V'_1, B - V'_2)$, $\alpha'_2 = e(V'_1, B + v'V'_2)$ and $\alpha'_3 = e(tA - v'V'_1, V'_2)$. Note that

$$\begin{array}{rcl} \alpha'_1 & = & e(A,B)^t e(A,-V'_2)^t e(-V'_1,B)e(V'_1,V'_2) \\ \alpha'_2 & = & e(V'_1,B)e(v'V'_1,V'_2) \\ \alpha'_3 & = & e(A,V'_2)^t e(-v'V'_1,V'_2) \end{array}$$

Thus, $o' = e(A,B)^t = \lambda'\alpha'_1\alpha'_2\alpha'_3$, where $\lambda' = e(-V'_1, V'_2)$.

Clearly, we have $o^t = o'$. Hence, **FPair** consists of the following steps for outsourcing the computation of bilinear pairings:

1. Run *Rand* twice to obtain two random tuples $(V_1, V_2, vV_1, vV_2, \lambda)$ and $(V'_1, V'_2, v'V'_1, v'V'_2, \lambda')$;
2. Query U_1 in random order as
   $U_1(A - V_1, B - V_2) \rightarrow \alpha_1$;
   $U_1(V'_1, B + v'V'_2) \rightarrow \alpha'_2$;
   $U_1(tA - v'V'_1, V'_2) \rightarrow \alpha'_3$;
   Similarly, query U_2 in random order as
   $U_2(tA - V'_1, B - V'_2) \rightarrow \alpha'_1$;
   $U_2(V_1, B + vV_2) \rightarrow \alpha_2$;
   $U_2(A - vV_1, V_2) \rightarrow \alpha_3$;
3. Compute $o = \lambda\alpha_1\alpha_2\alpha_3$ and $o' = \lambda'\alpha'_1\alpha'_2\alpha'_3$;
4. Finally, check whether or not both $o^t = o'$ and $o \in G_T$ holds. If it is the case, output o; otherwise, "error".

4.3 Proof

Theorem 3 *The algorithms (T, U_1, U_2) are an outsource-secure implementation of* **FPair** *in the TUP model, where the input (A, B) may be honest, secret; or honest, protected; or adversarial, protected.*

Proof. The correctness property is straight-forward, and we only prove the security. The proof is very similar to [35]. Let $\mathcal{A} = (E, U'_1, U'_2)$ be a probabilistic polynomial-time adversary that interacts with a probabilistic polynomial-time algorithm T in the one-malicious model.

Firstly, we prove *Pair One* $EVIEW_{real} \sim EVIEW_{ideal}$ (The external adversary, E, learns nothing.):

If the input (A, B) is anything other than honest, secret, then the simulator S_1 trivially behaves the same way as in the real execution.

If (A, B) is an honest, secret input, S_1 performs the following steps: In the i-th round, S_1 ignores the input and instead selects random points and $t \in \{1, \cdots, s\}$ for U_1 and U_2 to perform the random queries. S_1 checks the responses from U_1 and U_2. If there is an error, S_1 outputs $Y_p^i =$ "*error*", $Y_u^i = \phi$, $replace^i = 1$ and all states are saved. If there is no error, S_1 verifies the remaining outputs. If all checks pass, S_1 outputs $Y_p^i = \phi$, $Y_u^i = \phi$, $replace^i = 0$; otherwise, S_1 picks $o_r \in G_T$ randomly and outputs $Y_p^i = o_r$, $Y_u^i = \phi$, $replace^i = 1$. In either case, the appropriate states is saved by S_1.

Note that the input distributions to (U_1', U_2') in the ideal and real experiments are computationally indistinguishable. In the ideal experiment, the inputs are chosen randomly. In the real experiment, all numbers in the queries made by T is computationally indistinguishable from random since they are independently re-randomized.

If (U_1', U_2') is honest in the i-th round, $EVIEW_{real}^i \sim EVIEW_{ideal}^i$ (since $T^{(U_1', U_2')}$ executes **FPair** in the real experiment perfectly and S_1 simulates with the same outputs in the ideal experiment). If one of (U_1', U_2') behaves dishonestly in the i-th round and it is detected by T and S_1 (with probability $(1 - \frac{1}{3s})^2$), an output of "error" will be given; otherwise, the output of **FPair** will be successfully corrupted (with probability $1 - (1 - \frac{1}{3s})^2$). In the real experiment, the outputs generated by (U_1', U_2') are multiplied together along with a random value, thus a corrupted output of **FPair** looks random to E. In the ideal experiment, S_1 also simulates with a random value $o_r \in G_T$, which means that U_1 or(and) U_2 has(have) guessed the value t and the order of inputs, and returns wrong results. Thus, $EVIEW_{real}^i \sim EVIEW_{ideal}^i$ even when one of (U_1', U_2') is dishonest. Hence, we can conclude that $EVIEW_{real} \sim EVIEW_{ideal}$.

Secondly, we prove *Pair Two* $UVIEW_{real} \sim UVIEW_{ideal}$ (The untrusted software, (U_1', U_2'), learns nothing.):

The simulator S_2 always performs as follows: In the i-th round, S_2 ignores the input and instead selects random points and $t \in \{1, \cdots, s\}$ 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 since the output in the ideal experiment is never corrupted. However, this information cannot be communicated to (U_1', U_2') by E since T always re-randomizes its inputs to (U_1', U_2') in the i-th round of the real experiment. In the ideal experiment, S_2 always makes random, independent queries for (U_1', U_2'). Thus, for each round i we have $UVIEW_{real}^i \sim UVIEW_{ideal}^i$. Hence, we can conclude that $UVIEW_{real} \sim UVIEW_{ideal}$. □

Theorem 4 *The algorithms* $(T, (U_1, U_2))$ *are an* $(O(\frac{logs}{logq}), (1 - \frac{1}{3s})^2)$*-outsource-secure implementation of* **FPair**.

Proof. It takes the proposed algorithm **FPair** only 2 invocation of *Rand*, $O(logt)$ modular multiplications in G_T and $O(logt)$ point additions in G_1 (or G_2) in order to compute $e(A, B)$. Also, the computation for *Rand* is negligible since we use the table-lookup method. We then claim that the online computation is about $O(logs)$ since $t \in \{1, \cdots, s\}$. On the other hand, it takes roughly $O(logq)$ multiplications to compute a bilinear pair. Hence, the algorithms $(T, (U_1, U_2))$ are an $O(\frac{logs}{logq})$-efficient implementation of **FPair**.

By Theorem 3, U_1 (resp. U_2) cannot distinguish the test queries from the real queries T makes. Moreover, the only chance for U_1 or U_2 to keep the verification equation holding but output a faked value is to guess the value t (the reason is very similar to the proof of Algorithm B in [35]). If U_1 (resp. U_2) fails during any execution of **FPair**, it will be detected with probability $(1 - \frac{1}{3s})^2$. □

4.4 Efficiency

Here, we compare our proposal with Algorithm B [35]. The comparison of the efficiency is presented in Table 3. The comparison of pre-computation is shown in Table 4. Clearly, as for the efficiency of T, Algorithm B and our proposal are same, while the pre-computation of our proposal is superior to that of Algorithm B in efficiency.

Table 3: Efficiency Comparison

	Algorithm B [35]	Ours (**FPair**)
T	$O(logs)$ PA + $O(logs)$ M	$O(logs)$ PA + $O(logs)$ M
U	6P	6P
Tuple-size	10	10
Checkability	$(1 - \frac{1}{3s})^2$	$(1 - \frac{1}{3s})^2$

Table 4: Comparison of Pre-computations

	Algorithm B [35]	Ours
SM	6	4
Exp	2	2
M	8	2
MInv	4	-
PA	$2(k + h - 3)$	$4(k + h - 3)$

5 Conclusion

In this paper, we propose two new and efficient outsource-secure algorithm for bilinear pairings. One is more efficient than the state-of-the-art algorithms [18, 35] in both the computation of T and pre-computation. The other, which is flexible in the TUP model, is more efficient than the state-of-the-art algorithm [35] in pre-computation.

References

[1] M.J. Atallah and K.B. Frikken, “Securely Outsourcing Linear Algebra Computations,” in Proc. 5th ACM Symp. Inf.,Comput.Commun. Secur., 2010, pp. 48-59.

[2] M. Abadi, J. Feigenbaum, and J. Kilian, “On Hiding Information from an Oracle,” in Proc. 19th Annu. ACM Symp. Theory Comput., 1987, pp. 195-203.

[3] M.J. Atallah, K.N. Pantazopoulos, J.R. Rice, and E.H. Spafford, “Secure Outsourcing of Scientific Computations,” Adv. Comput., vol. 54, pp. 215-272, 2002.

[4] M.J. Atallah and J. Li, “Secure Outsourcing of Sequence Comparisons,” Int’l J. Inf. Secur., vol. 4, no. 4, pp. 277-287, Oct. 2005.

[5] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway, "Locally Random Reductions: Improvements and Applications," J. Cryptol., vol. 10, no. 1, pp. 17-36, Dec. 1997.

[6] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable Delegation of Computation Over Large Datasets," in Proc. Crypto, 2011, vol. LNCS 6841, pp. 111-131.

[7] D. Benjamin and M.J. Atallah, "Private and Cheating-Free Outsourcing of Algebraic Computations," in Proc. 6th Annu. Conf. Privacy, Secur. Trust, 2008, pp. 240-245.

[8] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson, "Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions," in Proc. ACM Symp. Theory Comput., 1988, pp. 113-131.

[9] M. Blanton, "Improved Conditional E-Payments," in Proc. ACNS, 2008, vol. LNCS 5037, pp. 188-206.

[10] M. Blanton, M.J. Atallah, K.B. Frikken, and Q. Malluhi, "Secure and Efficient Outsourcing of Sequence Comparisons," in Proc. ESORICS, 2012, vol. LNCS 7459, pp. 505-522.

[11] M. Blum, M. Luby, and R. Rubinfeld, "Program Result Checking Against Adaptive Programs and in Cryptographic Settings," Proc. DIMACS Series Discrete Math. Theoretical Comput. Sci., 1991, pp. 107-118.

[12] M. Blum, M. Luby, and R. Rubinfeld, "Self-Testing/Correcting with Applications to Numerical Problems," J. Comput. Syst. Sci., vol. 47, no. 3, pp. 549-595, Dec. 1993.

[13] V. Boyko, M. Peinado, and R. Venkatesan, "Speeding Up Discrete Log and Factoring Based Schemes via Precomputations," in Proc. Eurocrypt, 1998, vol. LNCS 1403, pp. 221-232.

[14] R. Canetti, B. Riva, and G. Rothblum, "Practical Delegation of Computation using Multiple Servers," in Proc. 18th ACM Conf. Comput. Commun. Secur., 2011, pp. 445-454.

[15] B. Carbunar and M. Tripunitara, "Conditioal Payments for Computing Markets," in Proc. CANS, 2008, vol. LNCS 5339, pp. 317-331.

[16] B. Carbunar and M. Tripunitara, "Fair Payments for Outsourced Computations," in Proc. SECON, 2010, pp. 529-537.

[17] D. Chaum and T. Pedersen, "Wallet Databases with Observers," in Proc. Crypto 1992, 1993, vol. LNCS 740, pp. 89-105.

[18] X. Chen, W. Susilo, J. Li, D.S. Wong, J. Ma, S. Tang and Q. Tang, "Efficient algorithms for secure outsourcing of bilinear pairings," Theoretical Computer Science, 2015, 562. pp. 112-121.

[19] B. Chevallier-Mames, J. Coron, N. McCullagh, D. Naccache, and M. Scott, "Secure Delegation of Elliptic-Curve Pairing," in Proc. CARDIS, 2010, vol. LNCS 6035, pp. 24-35.

[20] R. Cramer and V. Shoup, “Design and Analysis of Practical Public-Key Encryption Schemes Secure Against Adaptive Chosen Ciphertext Attack,” SIAMJ. Comput., vol. 33,no. 1,pp. 167-226, 2004.

[21] S. Even, O. Goldreich, and S. Micali, “On-Line/Off-Line Digital Signatures,” J. Cryptol., vol. 9, no. 1, pp. 35-67, 1996.

[22] R. Gennaro, C. Gentry, and B. Parno, “Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers,” in Proc. Crypto, 2010, vol. LNCS 6223, pp. 465-482.

[23] M. Girault and D. Lefranc, “Server-Aided Verification: Theory and Practice,” in Proc. ASIACRYPT, 2005, vol. LNCS 3788, pp. 605-623.

[24] S. Goldwasser, Y.T. Kalai, and G.N. Rothblum, “Delegating Computation: Interactive Proofs for Muggles,” in Proc. ACM Symp. Theory Comput., 2008, pp. 113-122.

[25] S. Goldwasser, S. Micali, and C. Rackoff, “The Knowledge Complexity of Interactive Proof-Systems,” SIAM J. Comput., vol. 18, no. 1, pp. 186-208, Feb. 1989.

[26] P. Golle and I. Mironov, “Uncheatable Distributed Computations,” in Proc. CT-RSA, 2001, vol. LNCS 2020, pp. 425-440.

[27] M. Green, S. Hohenberger, and B. Waters, “Outsourcing the Decryption of ABE Ciphertexts,” in Proc. 20th USENIX Conf. Secur., 2011.

[28] S. Hohenberger and A. Lysyanskaya, “How to Securely Outsource Cryptographic Computations,” in Proc. TCC, 2005, vol. LNCS 3378, pp. 264-282, Springer-Verlag: New York, NY, USA.

[29] T. Matsumoto, K. Kato, and H. Imai, “Speeding up Secret Computations with Insecure Auxiliary Devices,” in Proc. Crypto, 1988, vol. LNCS 403, pp. 497-506.

[30] P.Q. Nguyen, I.E. Shparlinski, and J. Stern, “Distribution of Modular Sums and the Security of Server-Aided Exponentiation,” in Proc. Workshop Comput. Number Theory Crypt., 1999, pp. 1-16.

[31] B. Parno, M. Raykova, and V. Vaikuntanathan, “How to Delegate and Verify in Public: Verifiable Computation from Attribute-Based Encryption,” in Proc. TCC, 2012, vol. LNCS 7194, pp. 422-439.

[32] K. Ren, C. Wang, and Q. Wang, “Security Challenges for the Public Cloud,” IEEE Internet Comput., vol. 16, no. 1, pp. 69-73. 2012.

[33] C.P. Schnorr, “Efficient Signature Generation for Smart Cards,” J. Cryptol., vol. 4, no. 3, pp. 161-174, 1991.

[34] L. Shi, B. Carbunar, and R. Sion, “Conditional E-Cash,” in Proc. FC, 2007, vol. LNCS 4886, pp. 15-28.

[35] H. Tian, F. Zhang, and K. Ren, “Secure Bilinear Pairing Outsourcing Made More Efficient and Flexible,” ASIA CCS’15, pp. 417-426, 2015.

[36] C. Wang, N. Cao, K. Ren, and W. Lou, “Enabling Secure and Efficient Ranked Keyword Search Over Outsourced Cloud Data,” IEEE Trans. Parallel Distrib. Syst., vol. 23,no. 8, pp.1467-1479, Aug. 2012.

[37] C. Wang, K. Ren, and J. Wang, “Secure and Practical Outsourcing of Linear Programming in Cloud Computing,” in Proc. 30th IEEE Int’l Conf. Comput. Commun., 2011, pp. 820-828.

[38] W. Wu, Y. Mu, W. Susilo, and X. Huang, “Server-Aided Verification Signatures: Definitions and New Constructions,” in Proc. ProvSec, 2008, vol. LNCS 5324, pp. 141-155.