Securely Outsourcing of Bilinear Pairings with Single Server

Abstract: Bilinear mappings are popular cryptographic primitives. It has been widely used in various modern cryptographic protocols. The computation of bilinear pairing is the most expensive operation in pair-based cryptographic schemes. At present, most of the algorithms for outsourcing bilinear pairing have small checkability or the outsourcers need to operate expensive computations with two servers. In this paper, we propose an efficient and secure algorithm for outsourcing bilinear pairing with single server, where the outsourcers can detect the errors with a probability of 1 if the cloud servers are dishonest, and some complex computations are not required by the outsourcers.

Keywords: elliptic curve;bilinear pairing; outsourcing computation

1 Introduction

With the development of cloud computing and the popularity of mobile devices, outsourcing computing has also attracted widespread attention. Outsourcing computing tasks to the cloud, the server can greatly reduce the users' computational cost. But the server is not completely trusted, it also meets many security challenges. To prevent untrusted cloud servers leaking and misusing users' data, users need to deliver to cloud server operations. Data is encrypted or blinded to ensure data privacy. In addition, users can verify the calculation results returned by the cloud server in order to obtain the correct calculation results. [1]

In the field of cryptography, outsourcing expensive calculations to semi-trusted devices has been extensively studied. Chaum et al. introduced the concept of "wallets with observers", installing hardware devices on users' hosts to perform complex operations. [2] Hohenberger et al. formally defined this model and proposed an outsource algorithm for modular exponentiation based on two servers that cannot collaborate with each other. [3] Chen et al. proposed an improved algorithm for modular exponentiation based on the same model. In this improved algorithm, the computational efficiency and verifiable probability of the outsourced algorithm are improved. [1] Green et al. proposed an outsourced decryption algorithm with attribute-based encryption (ABE) that reduces the user's decryption cost. However, the algorithm cannot verify the correctness of the outsourced results. [4] Lai et al. proposed a verifiable outsourced decryption algorithm in the ABE scheme that users can check the

outsourcing results. [5]

In the Cryptography, Bilinear pairs are widely used in many aspects. But the calculation of bilinear pairings is the most time-consuming and has an important influence on the efficiency of a solution or protocol. Therefore, a lot of work has been done on improving computational efficiency.

Chevallier-Mames et al. proposed an outsourced bilinear pairing scheme based on an untrusted server. If the server was dishonest, users can detect errors with a probability of 1. However, users need to perform some highly complex operations such as modular exponentiations. The complexity of these operations were equivalent to the calculation of bilinear pairings. [6] Subsequently, this problem has been found in the outsourcing of bilinear pairs [7, 8]. Chen et al. proposed the first practical bilinear pairing algorithm outsourcing algorithm based on two untrustworthy servers. The user only needs to perform five times of point addition and four times of modulus. The multiplication operation did not require any high-complexity operation and was suitable for the calculation of limited devices. However, the verifiable probability of the algorithm was only $1/2$, that is, the server can still deceive users with a probability of $1/2$. [9] Recently, Tian et al. proposed two bilinear operations based on two servers. One improved the efficiency of the outsourced algorithm. The user only need to perform 4 points plus 3 multiplications, but the verifiable probability is also 1/2. One increased the verifiable probability to $(1-1/3s)^2$, where s is a small integer, so the user may still be deceived by the server and accepted incorrect calculation result. [10] Min dong et al. proposed an efficient algorithm for bilinear pairings with two untrustworthy servers. In this model, the outsourcers can detect the errors with the probability of 1, but the cost of outsourcing to two servers was more than the cost of outsourcing to single server.

In our paper, we propose an efficient and secure algorithm of outsourcing bilinear pairings with single server. In this model, the outsourcers can detect the errors with probability of 1, and outsources never need to perform any complex computations at the same time. What's more, we can reduce the cost of outsourcing to the server.

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 $R \in G_1$, $Q \in G_2$.

2.2 Definition of Outsource-Security

Let Alg be a cryptographic algorithm. Simply speaking, an honest device T was outsourcing some work to the cloud server U securely, 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 not learn anything 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 [12] 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 $A=(E,U')$ 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 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 [12]:

Definition 2 (Outsource-Security) Let $A\text{lg}$ be an algorithm with outsource-I/O. A pair of algorithms (T,U) are an outsource-secure implementation of $A\text{lg}$ if:

1. Correctness: T^U is a correct implementation of $A\text{lg}$.
2. Security: For all probabilistic polynomial-time adversaries $\text{A}=(\text{E},\text{U}')$, there exist probabilistic expected polynomial-time simulators (S_1, S_2) such that the following pairs of random variables are computationally indistinguishable.

Pair One: $EVIEW_{real} \sim EVIEW_{ideal}$, which means that E get nothing useful about the inputs and outputs during the process of T^U.

The adversarial environment E gets the view by participating in the following real process:

$$EVIEW_{real}^{i} = \{(istate^i, x_{hs}^i, x_{hp}^i, x_{hu}^i) \leftarrow I(1^k, istate^{i-1});$$

$$(estate^i, j^i, x_{ap}^i, x_{au}^i, stop^i) \leftarrow E(1^k, EVIEW_{real}^{i-1}, x_{hp}^i, x_{hu}^i);$$

$$(tstate^i, ustate^i, y_s^i, y_p^i, y_u^i) \leftarrow T^{U'(ustate^{i-1})}$$

$$\times(tstate^{i-1}, x_{hs}^{j^i}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^i, x_{au}^i) : (estate^i, y_p^i, y_u^i)\}$$

$EVIEW_{real} = EVIEW_{ideal}$ 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 $(x_{hs}^i, x_{hp}^i, x_{hu}^i)$ 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_{ap}^i;

(d) the adversarial, unprotected input x_{au}^i;

(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 $\left(tstate^{i-1}, x_{hs}^{j^i}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^{i}, x_{au}^{i}\right)$, where $tstate^{i-1}$ is T's previously saved state, and produces a new state $tstate^{i}$ for T, as well as the secret y_s^i, protected y_p^i and unprotected y_u^i outputs. The oracle U' is given its previously saved state, $ustate^{i-1}$, as input, and the current state of U' is saved in the variable $ustate^{i}$. The view of the real process in round i consists of $estate^{i}$, and the values y_p^i and y_u^i. The overall view of E in the real process is just its view in the last round (i.e., i for which $stop^i = TRUE$.).

The ideal process:

$$EVIEW_{ideal}^{i} = \{\left(istate^{i}, x_{hs}^{i}, x_{hp}^{i}, x_{hu}^{i}\right) \leftarrow I\left(1^k, istate^{i-1}\right);$$

$$\left(estate^{i}, j^{i}, x_{ap}^{i}, x_{au}^{i}, stop^{i}\right) \leftarrow E\left(1^k, EVIEW_{ideal}^{i-1}, x_{hp}^{i}, x_{hu}^{i}\right);$$

$$\left(astate^{i}, y_s^i, y_p^i, y_u^i\right) \leftarrow A\mathrm{lg}\left(astate^{i-1}, x_{hs}^{j^i}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^{i}, x_{au}^{i}\right);$$

$$\left(sstate^{i}, ustate^{i}, Y_p^i, Y_u^i, replace^{i}\right)$$

$$\leftarrow S_1^{U'\left(ustate^{i-1}\right)} \times \left(sstate^{i-1}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^{i}, x_{au}^{i}, y_p^i, y_u^i\right);$$

$$\left(z_p^i, z_u^i\right) = replace^{i}\left(Y_p^i, Y_u^i\right) + \left(1 - replace^{i}\right)\left(y_p^i, y_u^i\right) : \left(estate^{i}, z_p^i, z_u^i\right)\}$$

The ideal process also proceeds in rounds. In the ideal process, there is a simulator S_1 who, shielded from the secret input x_{hs}^i, but given the non-secret outputs that $A\mathrm{lg}$ produces when run all the inputs for round i, decides to either output the values $\left(y_p^i, y_u^i\right)$ generated by $A\mathrm{lg}$, or replace them with some other values $\left(Y_p^i, Y_u^i\right)$. Note that the indicator variable $replace^{i}$ is a bit for determining whether y_p^i will be replaced with Y_p^i. Hence, it is allowed to query oracle U'; moreover, U' saves its state as in the real experiment.

Pair Two: $UVIEW_{real} \sim UVIEW_{ideal}$, i.e., the untrusted software U', which is written by E, will not know anything about inputs and outputs. The untrusted

software U' gets the view by participating in the real process which is described in **Pair One**. $UVIEW_{real} = ustate^i$ if $stop^i = TRUE$.

The ideal process:

$$UVIEW_{ideal}^i = \{\left(istate^i, x_{hs}^i, x_{hp}^i, x_{hu}^i\right) \leftarrow I\left(1^k, istate^{i-1}\right);$$

$$\left(estate^i, j^i, x_{ap}^i, x_{au}^i, stop^i\right) \leftarrow E\left(1^k, estate^{i-1}, x_{hp}^i, x_{hu}^i, y_p^i, y_u^i\right);$$

$$\left(astate^i, y_s^i, y_p^i, y_u^i\right) \leftarrow A\mathrm{lg}\left(astate^{i-1}, x_{hs}^{j^i}, x_{hp}^{j^i}, x_{hu}^{j^i}, x_{ap}^i, x_{au}^i\right);$$

$$\left(sstate^i, ustate^i\right) \leftarrow S_2^{U'\left(ustate^{i-1}\right)}\left(sstate^{i-1}, x_{hu}^{j^i}, x_{au}^i\right) : \left(ustate^i\right)\}$$

$UVIEW_{real} \sim UVIEW_{ideal}$ if $stop^i = TRUE$.

In the ideal process, we have a stateful simulator S_2 who, equipped with only the unprotected inputs $\left(x_{hu}^i, x_{au}^i\right)$, queries U'. As before, U' may maintain state.

Definition 3 (α-Efficient, Secure Outsourcing) The algorithms (T,U) are an α-efficient implementation of $A\mathrm{lg}$ if 1) T^U is a correct implementation of $A\mathrm{lg}$ and 2) $\forall$ inputs x, the running time of T is no more than an α-multiplicative factor of the running time of $A\mathrm{lg}$.

Definition 4 (β-Checkable, Secure Outsourcing) The algorithms (T,U) are a β-checkable implementation of $A\mathrm{lg}$ if 1) T^U is a correct implementation of $A\mathrm{lg}$ 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 $A\mathrm{lg}$ if they are both α-efficient and β-checkable.

3 New Outsourcing Algorithm of Bilinear Pairings

3.1 Security model

Hohenberger and Lysyanskaya first presented a model that is the so-called two untrusted program for outsourcing cryptographic computations.[12] In this model, even if we know that the returned result is incorrect, we cannot determine which server is dishonest. Hence, we propose a new secure outsourcing algorithm for bilinear pairings with single server.

Similar to[12], we also use a subroutine named *Rand* in order to speed up the computations. The inputs for *Rand* are the groups G_1 and G_2 with prime order q, the bilinear pairing e, and possibly some other (random) values, and the outputs for each invocation are a random, independent six-tuple $\left(V_1, V_2, \delta_1 V_1, \delta_2 V_1, \delta_2 V_2, e\left(\delta_1 V_1, \delta_2 V_2\right)\right)$, where $\delta_1, \delta_2 \in \mathbb{Z}_q^*$, $V_1 \in G_1$, and $V_2 \in G_2$. A naive way to achieve this functionality is to have a trusted server which compute a table of random, independent six-tuple in advance and then load it into the memory of T. For each invocation of *Rand*, T just retrieves a new six-tuple in the table (the table-lookup method).

3.2 Outsourcing algorithm

In this section, we propose a new efficient and secure bilinear pairings outsourcing algorithm. In our paper, T invokes the subroutine *Rand* to outsourcing its pairing computations to U. A requirement is that any useful information about the inputs and outputs cannot be known by the adversary A.

The input is two random points $A \in G_1$, $B \in G_2$, and the output is $e\left(A, B\right)$. Note that A and B may be secret or (honest/adversarial) protected and $e\left(A, B\right)$ is always secret or protected. Moreover, both A and B are computationally blinded to U. We let $U\left(\Lambda_1, \Lambda_2\right) \to e\left(\Lambda_1, \Lambda_2\right)$ denote that U takes as inputs $\left(\Lambda_1, \Lambda_2\right)$ and outputs $e\left(\Lambda_1, \Lambda_2\right)$. The proposed outsourcing algorithm contains the following steps:

(1) To achieve this functionality using U, T runs *Rand* to create two blinding six-tuples $\left(V_1, V_2, \delta_1 V_1, \delta_2 V_1, \delta_2 V_2, e\left(\delta_1 V_1, \delta_2 V_2\right)\right)$ and $\left(V_3, V_4, \delta_3 V_3, \delta_4 V_3, \delta_4 V_4, e\left(\delta_3 V_3, \delta_4 V_4\right)\right)$. We denote $\lambda_1 = e\left(\delta_1 V_1, \delta_2 V_2\right)$ and $\lambda_2 = e\left(\delta_3 V_3, \delta_4 V_4\right)$.

(2) The main skill of our paper is to logically split A and B into random looking pieces that can be computed by U. Without loss of generality, let $\alpha_1 = e\left(A + \delta_1 V_1, B + \delta_2 V_2\right)$, $\alpha_2 = e\left(\delta_2 A + \delta_2 V_1, V_2\right)$, $\alpha_3 = e\left(V_1, \delta_1\left(B + V_2\right)\right)$,

$\alpha_4 = e(\alpha_1 V_1 + \alpha_2 V_1, V_2)$, $\beta_1 = e(A + \delta_3 V_3, B + \delta_4 V_4)$, $\beta_2 = e(\delta_4 A + \delta_4 V_3, V_4)$,

$\beta_3 = e(V_3, \delta_3 (B + V_4))$, and $\beta_4 = e(\alpha_3 V_3 + \alpha_4 V_3, V_4)$.

Note that

$$\alpha_1 = e(A, B) e(A, \delta_2 V_2) e(\delta_1 V_1, B) e(\delta_1 V_1, \delta_2 V_2),$$

$$\alpha_2 = e(A, \delta_2 V_2) e(V_1, \delta_2 V_2),$$

$$\alpha_3 = e(\delta_1 V_1, B) e(\delta_1 V_1, V_2),$$

$$\alpha_4 = e(\delta_1 V_1, V_2) e(\delta_2 V_1, V_2),$$

$$\beta_1 = e(A, B) e(A, \delta_4 V_4) e(\delta_3 V_3, B) e(\delta_3 V_3, \delta_4 V_4),$$

$$\beta_2 = e(A, \delta_4 V_4) e(V_3, \delta_4 V_4),$$

$$\beta_3 = e(\delta_3 V_3, B) e(\delta_3 V_3, V_4),$$

$$\beta_4 = e(\delta_3 V_3, V_4) e(\delta_4 V_3, V_4).$$

Therefore, $e(A, B) = \alpha_1 \alpha_2^{-1} \alpha_3^{-1} \lambda_1^{-1} \alpha_4 = \beta_1 \beta_2^{-1} \beta_3^{-1} \lambda_2^{-1} \beta_4$.

(3) T queries U in random order as

$$U(A + \delta_1 V_1, B + \delta_2 V) \to \alpha_1,$$

$$U(\delta_2 A + \delta_2 V_1, V_2) \to \alpha_2,$$

$$U(V_1, \delta_1 (B + V_2)) \to \alpha_3,$$

$$U(\alpha_1 V_1 + \alpha_2 V_1, V_2) \to \alpha_4,$$

$$U(A + \delta_3 V_3, B + \delta_4 V_4) \to \beta_1,$$

$$U(\delta_4 A + \delta_4 V_3, V_4) \to \beta_2,$$

$$U(V_3, \delta_3 (B + V_4)) \to \beta_3,$$

$$U(\alpha_3 V_3 + \alpha_4 V_3, V_4) \to \beta_4.$$

(4) Finally, T checks that U produce the correct outputs, i.e., $\alpha_1 \alpha_2^{-1} \alpha_3^{-1} \lambda_1^{-1} \alpha_4$ and $\beta_1 \beta_2^{-1} \beta_3^{-1} \lambda_2^{-1} \beta_4$ for the test queries. If they are not equal, T outputs “error”;

otherwise, T can compute $e(A,B)=\alpha_1\alpha_2^{-1}\alpha_3^{-1}\lambda_1^{-1}\alpha_4$.

Remark 1 Given a random point P in G_1 (or G_2), T can compute the inverse point $-P$ easily.

Therefore, T can query

$$U(\delta_2 A+\delta_2 V_1,-V_2)\to e(\delta_2 A+\delta_2 V_1,-V_2)\to \alpha_2^{-1},$$

$$U(-V_1,\delta_1(B+V_2))\to e(-V_1,\delta_1(B+V_2))\to \alpha_3^{-1},$$

$$U(\delta_4 A+\delta_4 V_3,-V_4)\to e(\delta_4 A+\delta_4 V_3,-V_4)\to \beta_2^{-1},$$

$$U(-V_3,\delta_3(B+V_4))\to e(-V_3,\delta_3(B+V_4))\to \beta_3^{-1}.$$

Similarly, we can define the outputs of *Rand* be $\left(V_1,V_2,\delta_1V_1,\delta_2V_1,\delta_2V_2,e(\delta_1V_1,\delta_2V_2)^{-1}\right)$ and $\left(V_3,V_4,\delta_3V_3,\delta_4V_3,\delta_4V_4,e(\delta_3V_3,\delta_4V_4)^{-1}\right)$.

Hence, T needs not to express the inverse computation in G_T.

4 Security Proof

4.1 Security analysis

Theorem 1 The algorithms (T,U) are an outsource-secure implementation of our paper, where the input (A,B) may be honest, secret; or honest, protected; or adversarial, protected.

Proof The proof is similar to [12]. The correctness is insignificant and we only prove the security. Let $\mathrm{A}=(\mathrm{E},\mathrm{U}')$ be a PPT adversary that interacts with a PPT algorithm T.

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

Pay attention that we only consider three types of input (A,B): honest, secret; or honest, protected; or adversarial, protected. If the input (A,B) is anything other than honest, secret (this indicates that the input (A,B) is either honest, protected or adversarial, protected. Obviously, neither types of input (A,B) is secret), then the simulation is insignificant. In other words, the simulator S_1 behaves the same way in

the real execution. In general, S_1 never requires to access the secret input since neither types of input (A,B) is secret.

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

Distributing the input to $U^{'}$ in the real and ideal experiments are computationally indistinguishable. In the ideal environment, the inputs are randomly selected and distributed. In the real experiment, all numbers in the queries that T makes to the program in the step (3) of our paper is re-randomized independently and the re-randomization factors are also randomly generated by using naive table-lookup method. We consider the following case:

If $U^{'}$ is honest in the round i, the $EVIEW_{real}^i = EVIEW_{ideal}^i$ (this is because $T^{U^{'}}$ perfectly executes our paper in the real environment and S_1 simulates with the same outputs in the ideal environment, i.e., $rep^i = 0$). In the real environment, the four outputs generated by $U^{'}$ are multiplied together along with a random value λ^{-1} (see the step (4) of our algorithm). Thus, the output looks randomly to the environment E. In the ideal environment, S_1 also simulates with a random value $r \in G_T$ as the output. Thus, $EVIEW_{real}^i = EVIEW_{ideal}^i$. By the argument, we conclude

that $EVIEW_{real} = EVIEW_{ideal}$.

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

The behavior of the simulator S_2 is following: When the input is received on round i, S_2 ignores it, instead of making four random queries of the form (P_j, Q_j) to U'. Then S_2 saves its states and the states of U'. It is easy to distinguish between these real and ideal environment (note that the output of the ideal environment is never damaged). However, E cannot communicate this information to U'. This is because in the round i of the real environment, T always re-randomizes its inputs to U'. In the ideal environment, S_2 always generates random, independent queries for U'. Thus, for each round i we have $UVIEW^i_{real} \sim UVIEW^i_{ideal}$. By the argument, we conclude that $UVIEW_{real} \sim UVIEW_{ideal}$.

Theorem 2 The algorithms (T,U) are an $\left(O\left(\frac{1}{n}\right),1\right)$-outsource-secure implementation of our paper, where n is the bit length of the order q of bilinear groups.

Proof The proposed algorithm performs 2 calls to $Rand$ plus 10 point addition in G_1 (or G_2), 8 multiplication in G_T ,and 4 point multiplication in G_1 (or G_2) in order to compute $e(A,B)$. The computation for $Rand$ is negligible when we use the table-lookup method. On the other hand, it takes approximately $O(n)$ multiplications to compute the bilinear pairings in resulting finite filed. Thus, the algorithms (T,U) are an $O\left(\frac{1}{n}\right)$-efficient implementation of our paper.

On the other hand, T can check the returned queries from the real environment. If U fails during any execution of our paper, it will be detected with probability 1.

4.2 Comparison

We compare the proposed algorithm with the algorithm in [6]. We denote by PA a point addition in G_1 (or G_2), by SM a point multiplication in G_1 (or G_2), by M a

multiplication in G_T, by Inv an inverse in G_T, by Exp an exponentiation in G_T, and P a computation of the bilinear pairing. We omit other operations such as modular additions in $\mathbb{Z}_q$.

Table 1 shows the comparison among different outsourcing algorithms of bilinear pairings. Compared with the algorithm [6], the proposed algorithm is much superior in efficiency with the checkability. More precisely, compared with the algorithm [9, 10], this algorithm improves the checkability to 1. Moreover, compared with algorithm [11, 13], this proposed algorithm achieve outsourcing computing of bilinear pairings with single server. In this algorithm, little computational cost is appended.

On the other hand, it takes the servers U to perform 8P in our algorithm. In addition, the computation for *Rand* is about 2P + 2Exp + 6SM, which is negligible due to the table-lookup method. Therefore, the proposed algorithm requires more computation in the server side compared with [6]. However, note that the server is much more powerful in computing power, and thus the efficiency of our algorithm is not affected in this sense.

Table1. Comparison among different algorithms

Algorithm	BJN[6]	Pair[9]	TZR1[10]	TZR2[10]	VBP[13]	DBP[11]	Ours
PA(T)	4	5	4	$O(\log s)$	8	6	10
M(T)	6	4	3	$O(\log s)$	14	19	8
MExp(T)	10	0	0	0	0	0	0
SM(T)	6	0	0	0	0	0	4
Pair(U)	4	8	6	6	6	10	8
Servers	1	2	2	2	2	2	1
Checkability	1	$1/2$	$1/2$	$(1-1/3s)^2$	1	1	1

5 Conclusion

In this paper, we propose an efficient and secure outsourcing algorithm for bilinear pairings with single server. A distinguishing property of our proposed algorithm is that the outsourcer never requires to accomplish some expensive operations such as exponentiations and the outsourcers can detect the errors with a probability of 1 if the cloud servers are dishonest.

References

[1] Chen X F, Li J, Ma J F, et al. New algorithms for secure outsourcing of modular exponentiations. IEEE Trans Parall Distrib Syst, 2014, 25: 2386–2396.

[2] Chaum D, Pedersen T. Wallet databases with observers. Proceedings of 12th Annual Conference on Advances in Cryptology. Berlin: Springer, 1992. 89–105.
[3] Hohenberger S, Lysyanskaya A. How to securely outsource cryptographic computations. Proceedings of the 2nd International Conference on Theory of Cryptography. Berlin: Springer, 2005. 264–282.
[4] Green M, Hohenberger S, Waters B. Outsourcing the decryption of ABE ciphertexts. Proceedings of the 20th USENIX Conference on Security. New York: ACM, 2011. 34.
[5] Lai J Z, Deng R H, Guan C W, et al. Attribute-based encryption with verifiable outsourced decryption. IEEE Trans Inf Foren Secur, 2013, 8: 1343–1354.
[6] Chevallier-Mames B, Coron J, McCullagh N, et al. Secure delegation of elliptic-curve pairing. Proceedings of the 9th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Application. Berlin: Springer, 2010. 24–35.
[7] Tsang P, Chow S, Smith S. Batch pairing delegation. Proceedings of the 2nd International Workshop on Security. Berlin: Springer, 2007. 74–90.
[8] Chow S, Au M, Susilo W. Server-aided signatures verification secure against collusion attack. Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. New York: ACM, 2011. 401–405.
[9] Chen X F, Susilo W, Li J, et al. Efficient algorithms for secure outsourcing of bilinear pairings. Theor Comput Sci, 2015, 562: 112–121.
[10] Tian H B, Zhang F G, Ren K. Secure bilinear pairing outsourcing made more efficient and flexible. Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security. New York: ACM, 2015. 417–426.
[11]Min dong, Yanli Ren, Xinpeng Zhang. Fully Verifiable Algorithm for Secure Outsourcing of Blinear Pairing in Cloud Computing. KSII Transactions on Internet and Information Systems, 2017,7(11): 3648-3663.
[12]S. Hohenberger, A. Lysyanskaya, How to Securely Outsource Cryptographic Computations.in Proc. TCC, 2005, vol. LNCS 3378, pp. 264-282, Springer-Verlag: New York, NY, USA.
[13] Y. Ren, N. Ding, T. Wang et al., “New algorithms for verifiable outsourcing of bilinear pairing,” Science China Information Sciences, vol.59:099103, 2016.