Secure Outsourcing of Scalar Multiplication on Elliptic Curves

Kai Zhou and Jian Ren
Department of ECE, Michigan State University, East Lansing, MI 48824-1226
Email: {zhoukai, renjian}@msu.edu

***Abstract*—Cloud computing enables resource-constrained end-users to outsource their computational tasks to the cloud in a flexible manner. One major concern of computation outsourcing is the security of the outsourced data as well as the results. In this paper, we propose a secure outsourcing scheme (SecMul) for one basic and expensive computation in cryptography, that is scalar multiplication of points on elliptic curves. The basic idea of the proposed SecMul is to transfer computations in a finite field to computations in a ring. The scheme is designed in such a way that without the secret key, it is computationally infeasible for the cloud to recover the input and the output. The proposed SecMul is highly efficient in that it enables the end-users to outsource a scalar multiplication at the cost of only a few multiplications. Especially, the performance gain of outsourcing is $\mathcal{O}(\log(s))$, where s is the multiplier. Our comprehensive security and complexity analysis demonstrate that the proposed SecMul scheme is both secure and efficient.**

***Index Terms*—Cloud computing, secure outsourcing, scalar multiplication, elliptic curve**

I. Introduction

Computation outsourcing is an integral part of the services provided by cloud computing. In the computation outsourcing paradigm, the cloud servers are regarded as a shared pool of resources such as storage and computational power. The resource-constrained end-users can outsource their computational tasks to the cloud in a pay-per-use manner. Despite the tremendous benefits provided by computation outsourcing, one major concern is the security of the outsourced data. When end-users' data is outsourced to the cloud, the data as well as the private information revealed by the data is at a risk especially when the data is sensitive. Thus, it is crucial to ensure the security of the outsourced data and the returned results in computation outsourcing.

To deal with the above issue, researchers have proposed different secure outsourcing schemes for various computational problems [1]–[5]. In the domain of cryptographic computations, previous works largely focus on outsourcing of modular exponentiation which is a basic computation in cryptography. The main idea of these works [3], [6] is to split the components of modular exponentiation into seemingly random pieces that are then outsource to two independent servers. One disadvantage of this technique is that the two independent servers have to be non-colluding; otherwise, the servers can easily recover the original outsourced data. Moreover, the above schemes rely heavily on pre-computations that may degrade the efficiency of the outsourcing process.

In this paper, we initiate the secure outsourcing of another basic computation in cryptography: scalar multiplication on elliptic curves. Elliptic curve cryptography [7] relies on the computation of scalar multiplication aP, where a is an integer and P is a point on an elliptic curve. There exists no explicit function to calculate scalar multiplication and the typical methods are to add the point repeatedly. Since cryptographic applications require the integer a to be large, scalar multiplication is computationally expensive thus motivating us to outsource scalar multiplication to the cloud. The main idea of our proposed outsourcing (SecMul) scheme can be illustrated as a decomposition and rebuilt approach. To be specific, we utilize the typical "double-and-add" algorithm [8, Chapter 5.3] to compute scalar multiplication. In this algorithm, the two basic building blocks are point addition and point doubling that consist of a set of basic operations: addition, multiplication and inversion in a finite field. We utilize the projective coordinates to avoid the inversion operation. Then, we propose a ring homomorphism to enable secure computation of addition and multiplication based on which we construct the procedure to compute point addition and point doubling. At last, based on the secure computation of the two building blocks, we can construct SecMul to securely outsource scalar multiplications. The main features and main contributions of our proposed SecMul can be summarized as follows:

- We initiate the secure outsourcing of scalar multiplication on elliptic curves.
- The proposed SecMul outsources a scalar multiplication to only one independent server that eliminates the non-colluding assumption.
- The proposed SecMul is highly efficient. It does not rely on pre-computations and only takes a few multiplications at local environment to outsource the problem and recover the result.

The rest of this paper is organized as follows. In Section II, we formally state the problem of secure outsourcing of scalar multiplication. In Section III, we propose our secure outsourcing scheme SecMul. The security and complexity analysis are presented in Section IV. We give the numeric results in Section V. At last, we conclude in Section VI.

II. Problem Statement

A. System Model and Threat Model

In the general settings of computation outsourcing, the system consists of two entities: an end-user E and the cloud

S. The cloud S is regarded as possessing unlimited resources and is able to carry out extensive computations. The end-user E is resource-constrained, that is it has limited computational power and storage space. It desires to accomplish a computational expensive problem $F(\mathbf{x}) \to w$, where $\mathbf{x}$ is the input and w is the output of the task. Due to the limited resources, E outsources the problem $F(\mathbf{x})$ to S. Unfortunately, the cloud cannot be fully trusted. Therefore, the basic requirement is that it would be infeasible for the cloud to derive any key information about $\mathbf{x}$ and w from the outsourced task.

In this paper, we assume the cloud S is *honest but curious*. That is, the cloud will honestly fulfill its advertised functionality. However, S is curious. It may try to derive any key information from the outsourced task, the input and the output as well as the intermediate computational process. When the outsourced data is sensitive, this will cause severe security and privacy issues.

B. Secure Outsourcing Scheme

We formally define a Secure Outsourcing Scheme (SOS) as a 3-tuple $(\mathcal{T}, \mathcal{C}, \mathcal{R})$ consisting of three functions.

1) **Problem Transformation** $\mathcal{T} : F(\mathbf{x}) \to G(\mathbf{y})$. The end-user E locally transforms the problem $F(\mathbf{x})$ to a new form $G(\mathbf{y})$, where $\mathbf{y}$ is the new input and G is the new problem description. E then outsources $G(\mathbf{y})$ to the cloud server S.
2) **Cloud Computation** $\mathcal{C} : G(\mathbf{y}) \to W$. The cloud S solves the transformed problem $G(\mathbf{y})$ to obtain the corresponding result W.
3) **Result Recovery** $\mathcal{R} : W \to w$. Based on the returned result W, the end-user E recovers the result w of the original problem $F(\mathbf{x})$.

We define the security of an outsourcing scheme as in Definition 1.

Definition 1 (**Security**). An outsourcing scheme is secure if given the outsourced problem $G(\mathbf{y})$, it is computationally infeasible to recover the original problem $F(\mathbf{x})$ and the result w.

To measure the performance of a SOS, we adopt a similar definition of efficiency as proposed in [3].

Definition 2 (**α-efficient**). Suppose the running time of a problem $F(\cdot)$ for E is t_0. Under an SOS, the running time of local processing for E is t_p. Then the SOS is α-efficient if $\frac{t_0}{t_p} \geq \alpha$.

From the definition above, we can see that a larger α indicates a better performance of a secure outsourcing scheme.

C. A Top-down View of Scalar Multiplication

We follow the definition of elliptic curves in [8, Chapter 5] and [9, Chapter 13]. An elliptic curve E over a field $\mathbb{K}$ is the set of solutions to the Weierstrass equation

$$E: \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

where the coefficients $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$ and each solution points (x, y) has its coordinates in the extension field $\overline{\mathbb{K}}$. Typically, in elliptic curve cryptography, we consider the short Weierstrass equation

$$E: \ y^2 = x^3 + a_4x + a_6, \tag{1}$$

where the coefficients a_4, a_6 and the coordinates of the points are all in a finite field $\mathbb{F}_p$. The elliptic curve defined above is denoted as $E(\mathbb{F}_p)$. Furthermore, for cryptographic applications, we usually work with points with large prime order that is defined as follows:

Definition 3 (**Torsion Points**). Suppose $m \geq 1$ is an integer. A point $P \in E(\mathbb{F}_p)$ is a m-torsion point if $mP = \mathcal{O}$, where $\mathcal{O}$ is the point at infinity. The set of m-torsion points on $E(\mathbb{F}_p)$ is denoted by

$$E(F_p)[m] = \{P \in E(F_p) : [m]P = \mathcal{O}\},$$

where $[m]$ is the equivalent class of m in $\mathbb{F}_p$.

The set of m-torsion points forms a subgroup of $E(\mathbb{F}_p)$ where each point has an order of m. In this paper, we focus on secure outsourcing of scalar multiplication that is defined as follows:

Definition 4 (**Scalar Multiplication**). For a point $P \in E(\mathbb{F}_p)[m]$ and an integer $a \in \mathbb{F}_p$, scalar multiplication is to compute aP, meaning to add P to itself a times.

The following "double-and-add" algorithm [8, Chapter 5.3] introduces a method to compute scalar multiplication. We can see that two basic building blocks utilized in this algorithm are point addition $(R \leftarrow R + Q)$ and point doubling $(Q \leftarrow 2Q)$. Given two points $P = (x_1, y_2)$ and $Q = (x_2, y_2)$ on an elliptic curve $E = \{p, a_4, a_6\}$, the functions to calculate point addition and point doubling are introduced in [9, Chapter 13] to which we refer the readers for the details. What we should notice is that one point addition $(P \neq \pm Q)$ takes 3 multiplications and 1 inversion in a finite field $\mathbb{F}_p$. When $P = Q$, one point doubling takes 4 multiplications and 1 inversion. To avoid inversion which is more expensive than multiplication, we utilize projective coordinates in the calculation of point addition and doubling.

Algorithm 1 Double-and-Add Algorithm to Compute Scalar Multiplication

```
Input: point $P \in E(\mathbb{F}_p)[m]$ and integer $a \in \mathbb{F}_p$.
Output: point $R = aP$.
1: Set $Q \leftarrow P$ and $R \leftarrow \mathcal{O}$.
2: while $(a > 0)$ do
3:     if $n \bmod 2 = 1$ then
4:         Set $R \leftarrow R + Q$.
5:     else
6:         Set $Q \leftarrow 2Q$ and $a \leftarrow \lfloor a/2 \rfloor$.
7:     end if
8: end while
9: Return the point $R = aP$.
```

In projective coordinates, a point $P = (x, y, z)$ corresponds to the point $Q = (\frac{x}{z}, \frac{y}{z})$ in the affine coordinates and the short Weierstrass equation (1) is transformed to

$$y^2 z = x^3 + a_4 x z^2 + a_6 z^3.$$

Accordingly, the point addition and point doubling in affine coordinates can be transformed to those in projective coordinates [9, Chapter 13]. Given two points $P = (x_1, y_1, z_1)$ and $Q = (x_{2,} y_2, z_2)$ such that $P \neq \pm Q$, the point addition $P + Q = (x_3, y_3, z_3)$ can be calculated as

$$x_3 = BC, y_3 = A(B^2 x_1 z_2 - C) - B^3 y_1 z_2, z_3 = B^3 z_1 z_2,$$

where

$$\begin{array}{l} A = y_2 z_1 - y_1 z_2, B = x_2 z_1 - x_1 z_2, \\ C = A^2 z_1 z_2 - B^3 - 2B^2 x_1 z_2. \end{array}$$

The point doubling $2P = (x_3, y_3, z_3)$ can be calculated as

$$x_3 = 2BD, y_3 = A(4C - D) - 8y_1^2 B^2, z_3 = 8B^3,$$

where

$$A = a_4 z_1^2 + 3x_1^2, B = y_1 z_1, C = x_1 y_1 B, D = A^2 - 8C.$$

In projective coordinates, one point addition and doubling take 14 multiplications and 12 multiplications, respectively.

Now the problem of secure outsourcing of scalar multiplication is clear. We want to outsource the given point P on an elliptic curve and an integer a to the cloud, expecting the cloud to return a result from which we can recover the desired scalar multiplication aP. In this outsourcing process, it should be infeasible for the cloud to derive any key information about the point P, the integer a and the result aP.

III. SECMUL: A BOTTOM-UP APPROACH

In the previous section, we introduce a top-down view of scalar multiplication. To be specific, the scalar multiplication can be decomposed to repeated point addition and point doubling. Further, by utilizing projective coordinates, we limit the basic operations to addition and multiplication. In this section, we introduce a bottom-up approach to securely outsource scalar multiplication. We first introduce a ring homomorphism and its polynomial-homomorphic property that enables secure computation of addition and multiplication. Based on this, we construct the scheme to securely compute point addition and point doubling from which we propose the secure outsourcing scheme for scalar multiplication.

A. Ring Homomorphism

Consider two rings and their corresponding operations $(\mathbb{R}_1, +, \cdot)$ and $(\mathbb{R}_2, \circ, \star)$ and a mapping function $f : \mathbb{R}_1 \to \mathbb{R}_2$. We define ring homomorphism as follows:

Definition 5 (Ring Homomorphism). Given $(\mathbb{R}_1, +, \cdot)$ and $(\mathbb{R}_2, \circ, \star)$, a mapping function $f : \mathbb{R}_1 \to \mathbb{R}_2$ is a ring homomorphism if there exists an inverse mapping function $g : \mathbb{R}_2 \to \mathbb{R}_1$ and the pair (f, g) possesses the following two properties:

- **Additive Homomorphic**: $\forall x_1, x_2 \in \mathbb{R}_1$, $x_1 + x_2 = g(f(x_1) \circ f(x_2))$;
  - **Multiplicative Homomorphic**: $\forall x_1, x_2 \in \mathbb{R}_1$, $x_1 \cdot x_2 = g(f(x_1) \star f(x_2))$.

The computations of scalar multiplication are operated in a finite filed $\mathbb{F}_p$ that is also a ring. Thus, our primitive goal is to construct a proper ring homomorphism f that maps elements in $\mathbb{F}_p$ to elements in a ring $\mathbb{R}$. In this way, the computations in $\mathbb{F}_p$ can be transformed to corresponding computations in $\mathbb{R}$ and the result in $\mathbb{R}$ can be further recovered in $\mathbb{F}_p$. We propose a ring homomorphism f as follows:

$$\begin{array}{cccc} f: & \mathbb{F}_p & \longrightarrow & \mathbb{R}_N \\ & x & \longmapsto & (x + kp) \bmod N, \end{array}$$

where k is a random integer in $\mathbb{F}_p$, $N = pq$ and p, q are large primes. Then we prove that f is a ring homomorphism.

Lemma 1. *For x, y and z in a ring $\mathbb{R}$, if $z \mid y$, then we have* $x \bmod y \bmod z = x \bmod z$.

Proof. The proof is straight forward. We can always write x in the form $x = sy + t$, where $s, t \in \mathbb{R}$. Then

$$x \bmod y \bmod z = t \bmod z.$$

When $z \mid y$, we have

$$x \bmod z = sy \bmod z + t \bmod z = t \bmod z.$$

Thus, $x \bmod y \bmod z = x \bmod z$. □

Theorem 1. *$\forall x \in \mathbb{F}_p$, the proposed mapping function $f(x) = (x + kp) \bmod N$ is a ring homomorphism.*

Proof. We show that there exists an inverse mapping function g and the pair (f, g) possesses the additive homomorphic property and multiplicative homomorphic property. The inverse mapping function g is give by

$$\begin{array}{cccc} g: & \mathbb{R}_N & \longrightarrow & \mathbb{F}_p \\ & y & \longmapsto & y \bmod p. \end{array}$$

Suppose $x_1, x_2 \in \mathbb{F}_p$ and $f(x_1) = (x_1 + k_1 p) \bmod N$ and $f(x_2) = (x_2 + k_2 p) \bmod N$, where $k_1, k_2 \in \mathbb{F}_p$ are random integers. Then we can verify that

$$\begin{array}{cl} & g(f(x_1) + f(x_2)) \\ = & ((x_1 + k_1 p) \bmod N + (x_2 + k_2 p) \bmod N) \bmod p \\ = & (x_1 + k_1 p + x_2 + k_2 p) \bmod N \bmod p \\ = & (x_1 + k_1 p + x_2 + k_2 p) \bmod p \\ = & (x_1 + x_2) \bmod p. \end{array} .$$

The third equality results from Lemma 1. Thus, we have proved that (f, g) has additive homomorphic property. Similarly, we can verify that (f, g) is also multiplicative homomorphic as follows:

$$\begin{array}{cl} & g(f(x_1) \cdot f(x_2)) \\ = & ((x_1 + k_1 p) \bmod N \cdot (x_2 + k_2 p) \bmod N) \bmod p \\ = & ((x_1 + k_1 p) \cdot (x_2 + k_2 p)) \bmod N \bmod p \\ = & ((x_1 + k_1 p) \cdot (x_2 + k_2 p)) \bmod p \\ = & x_1 \cdot x_2 \bmod p. \end{array} .$$

Hence, the proposed mapping function $f(x) = (x + kp) \bmod N$ is a ring homomorphism. □

The above proposed ring homomorphism enables us to transform the addition and multiplication in a finite field into the corresponding operations in a ring. We further explore the polynomial-homomorphic property of the ring homomorphism that is defined as follows.

Definition 6 (**Polynomial-Homomorphic**). Suppose $\mathbf{x} = (x_1, x_2, \cdots, x_n) \in \mathbb{F}_p^n$ and $\text{poly}(\mathbf{x})$ is a polynomial function defined on $\mathbf{x}$. A mapping function $f: \mathbb{F}_p \longrightarrow \mathbb{R}_N$ is polynomial-homomorphic if there exists an inverse mapping function $g: \mathbb{R}_N \longrightarrow \mathbb{F}_p$ such that

$$g(\text{poly}(f(\mathbf{x}))) = \text{poly}(\mathbf{x}),$$

where f is applied on $\mathbf{x}$ component-wise.

Theorem 2. *The proposed ring homomorphism $f(x) = (x + kp) \bmod N$ is polynomial-homomorphic.*

The proof of the above theorem is straightforward given the additive homomorphic and multiplicative homomorphic property of the ring homomorphism.

B. Secure Outsourcing of Scalar Multiplication

Theorem 2 states that by mapping the variables of a polynomial from a finite field to corresponding variables in a ring, we can evaluate the polynomial in the ring and recover the result in the finite field. This gives us the insight of our proposed SecMul since essentially, point addition and point doubling are both the process of evaluating polynomials on the coordinates of the points. Thus, we can construct the secure computation scheme for point addition and point doubling as in Algorithm 2.

Algorithm 2 Secure Point Addition and Point Doubling

```
Input: $P = (x_1, y_1, z_1)$, $Q = (x_2, y_2, z_2)$ and $E = \{a_4, a_6, p\}$
Output: point $R = P + Q = (x_3, y_3, z_3)$.
1: Select a large prime $q$ and compute $N = pq$.
2: For a coordinate $x_i$, select a random integer $k_i$ and
   compute $x'_i = (x_i + k_i p) \bmod N$.
3: Transorm the points $P, Q$ and the elliptic curve $E$ to
   $P' = (x'_1, y'_1, z'_1)$, $Q' = (x'_2, y'_2, z'_2)$ and $E' = \{a'_4, a'_6, N\}$
   respectively as described in Step 2.
4: Outsource $P', Q'$ and $E'$ to the cloud.
5: Cloud computes $R' = P' + Q'$ following the point
   doubling or point addition prodecure.
6: On receiving $R' = (x'_3, y'_3, z'_3)$, recover $R$ as $R =
   (x'_3, y'_3, z'_3) \bmod p = (x_3, y_3, z_3)$.
```

Theorem 3. *The proposed secure point addition and point doubling algorithm is sound.*

The proof of Theorem 3 is straightforward from the polynomial-homomorphic property of the ring homomorphism.

The above theorem enables us to conceal the points as well as the parameters of the elliptic curve from the cloud. To outsource scalar multiplication aP, the remaining part to conceal is the multiplier a. We utilize the property of the order m of the torsion group that is $rmP = \mathcal{O}$, for an arbitrary point $P \in E[m](\mathbb{F}_p)$ and any integer r. As a result, we can conceal s by adding it to a multiple of m as $s' = s + rm$, where r is a random integer. Now, we can summarize the proposed SecMul as in Algorithm 3.

Algorithm 3 Secure Outsourcing of Scalar Multiplication

```
Input: $P = (x_1, y_1, z_1)$, $s$, $E = \{a_4, a_6, p\}$ and $m$.
Output: point $R = sP$.
Key Generation:
  1: End-user selects a large prime $q$ and compute $N = pq$.
Problem Transformation:
  1: End-user generates random integers $k_1, k_2, k_3, k_4, k_6, r$.
  2: Computes $x'_1 = (x_1 + k_1 p) \bmod N$, $y'_1 = (y_1 + k_2 p) \bmod$
     $N$, $z'_1 = (z_1 + k_3 p) \bmod N$, $a = (a_4 + k_4 p) \bmod N$,
     $a'_6 = (a_6 + k_6 p) \bmod N$, $s' = s + rm$.
  3: End-user outsources $P' = (x'_1, y'_1, z'_1)$, $E' = (a'_4, a'_6, N)$
     and $s'$.
Cloud Computation:
  1: The cloud computes $R' = s'P'$ utilizing the double-and-
     add algorithm.
Result Recovery:
  1: The end-user recovers the result $R$ as $R =$
     $(x'_3, y'_3, z'_3) \bmod p$.
```

Theorem 4. *The propose SecMul is sound.*

Proof. From Theorem 3, we know that the secure computation scheme for point addition and point doubling is sound. Since the double-and-add algorithm to compute scalar multiplication consists of a series of point addition and point doubling, we have $R = s'P = (s + rm)P = sP + rmP = sP + \mathcal{O} = sP$. □

IV. Security and Complexity Analysis

In this section, we analyze the security and complexity of the proposed SecMul.

A. Security Analysis

The security of the proposed SecMul is based on the hardness of integer factorization. That is, given an integer $N = pq$, where p and q are two distinct large primes, it is computationally infeasible to factorize N to obtain p and q. Based on the security definition of an outsourcing scheme in Definition 1, we prove that our proposed SecMul is secure.

Theorem 5. *For the proposed SecMul scheme, it is computationally infeasible to recover $\mathbf{x}$ and w.*

Proof. The original input is $\mathbf{x} = \{P = (x_1, y_1, z_1), E = \{a_4, a_6, p\}, s\}$ and the transformed input is $\mathbf{y} = \{P' = (x'_1, y'_1, z'_1), E' = \{a'_4, a'_6, N\}, s'\}$. The output to the transformed problem $G(\mathbf{y})$ is $W = \{R' = (x'_3, y'_3, z'_3)\}$ and the output to the original problem $F(\mathbf{x})$ is $w = \{R = (x_3, y_3, z_3)\}$. We need to prove that given $\mathbf{y}$ and W, it is computational infeasible to recover $\mathbf{x}$ and w.

First, given $N = pq$, it is computationally infeasible to recover p since integer factorization is hard. Second, consider $x_1' = (x_1 + k_1 p) \bmod N$. Since p is secret and k_1 is randomly chosen, it is computationally infeasible to recover x_1 given x_1'. By the same argument, we can prove that it is also computational infeasible to recover y_1, z_1, a_4, a_6 and s. Thus we have proved that given $\mathbf{y}$ and W, it is computational infeasible to recover $\mathbf{x}$ and w. $\square$

B. Complexity Analysis

To show the efficiency of our proposed SecMul, we need to analyze the computational complexity of scalar multiplication, the problem transformation and the result recovery process. The most expensive basic operation involved in the computation is multiplication in a finite field. Thus, we utilize the number of multiplications as a measurement of the computational complexity.

In Algorithm 1, the "double-and-add" algorithm requires u point doubling to calculate the scalar multiplication sP, where u is the bit length of s and $u = \log s$. It also requires v point additions, where $0 < v < u$ and v is determined by the number of 1's in s. In Section II-C, we show that one point addition and doubling take 14 multiplications and 12 multiplications, respectively. Thus, to calculate sP, the number of multiplications required is $M_0 = 12u + 14v$. In our proposed SecMul, we need to transform the input $\mathbf{x}$ to $\mathbf{y}$ and recover the result w from W. In this two processes, the number of multiplications required is $M = 7$. Thus, the computational gain is $\tau = M_0/M = \frac{12u+14v}{7} = \theta \log(s)$, where θ is an efficiency parameter and is in the range of $[\frac{12}{7}, \frac{14}{7}]$.

From the above analysis, we have the following theorem stating the efficiency of SecMul.

Theorem 6. *The proposed SecMul is $\theta \log(s)$-efficient.*

V. Numeric Results

In this section, we measure the performance of SecMul through simulation. The computation of both the end-user and the cloud server is simulated using the same computer with an Intel Core 2 Due CPU running at 2.53 GHz with 4GB RAM. The multiple-precision computation is implemented based on the *$C++$ Multiple Precision Integers and Rationals* (MPIR) library. We denote t_p as the time for local processing in outsourcing, which includes problem transformation and result recovery. We let t_0 be the time for the end-user to calculate scalar multiplication sP if it is not outsourced to the cloud. Then, the performance gain is defined as $\tau = \frac{t_p}{t_0}$. Let b be the bit length of p, where $\mathbb{F}_p$ is the finite filed we operate in. In our simulation, we let b vary from 32 bits from 256 bits, since the key size of elliptic curve cryptography is typically hundreds of bits. We specify s as an integer of b bits and for each round, we randomly generate an s. The simulation is repeated 10^4 rounds for each fixed bit length b. The numeric result is shown in Table I. From the table, we can see that the performance gain is approximately $\theta \log(s)$, which conforms to our theoretical analysis. In our previous complexity analysis, we take the number of multiplications as a measurement. However, scalar multiplication also involves addition and modular operations besides multiplication. This explains the variation of the parameter θ in our numeric result.

Table I
Performance Gain of SecMul for Different Bit Length

Bit length b	t_p(ms)	t_0(ms)	Gain τ	θ
32	49.76	3357	67.5	2.1
48	51.48	6594	128.1	2.7
64	52.88	9485	179.4	2.8
80	54.14	12908	238.4	3.0
96	56.01	17371	310.1	3.2
112	56.78	22104	389.3	3.5
128	59.59	26786	449.5	3.5
144	60.22	30820	511.8	3.6
160	64.27	37546	584.2	3.7
176	65.32	43361	663.8	3.8
192	68.01	50063	736.1	3.8
208	69.11	59451	860.2	4.1
224	72.85	68973	946.8	4.2
240	74.10	79736	1076.1	4.5
256	78.16	86853	1111.2	4.3

VI. Conclusion

In this paper, we investigate the problem of secure outsourcing of scalar multiplication on elliptic curves, which is a basic operation in cryptography. In our proposed SecMul, we first view scalar multiplication as a series of point addition and doubling, which can be further decomposed into addition and multiplication in a finite field. We propose a ring homomorphism that enables secure computation of addition and multiplication. As a result, when the problem is outsourced to the cloud, it is computational infeasible for the cloud to recover the input and the output. Our proposed SecMul is sound in that the end-user can recover the correct result from the result returned by the cloud. The complexity analysis shows that SecMul can introduce a performance gain of $\mathcal{O}(\log(s))$.

References

[1] Mikhail J Atallah and Keith B Frikken. Securely outsourcing linear algebra computations. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 48–59. ACM, 2010.

[2] Marina Blanton, Mikhail J Atallah, Keith B Frikken, and Qutaibah Malluhi. Secure and efficient outsourcing of sequence comparisons. In *Computer Security–ESORICS 2012*, pages 505–522. Springer, 2012.

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

[4] Yujue Wang, Qianhong Wu, Duncan S Wong, Bo Qin, Sherman SM Chow, Zhen Liu, and Xiao Tan. Securely outsourcing exponentiations with single untrusted program for cloud storage. In *Computer Security-ESORICS 2014*, pages 326–343. Springer, 2014.

[5] 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*, 562:112–121, 2015.

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

[7] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[8] Jeffrey Hoffstein, Jill Catherine Pipher, Joseph H Silverman, and Joseph H Silverman. *An introduction to mathematical cryptography*. Springer, 2008.

[9] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC press, 2005.