Yang B, Yu Y, Yang CH. A secure scalar product protocol against malicious adversaries. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 28(1): 152–158 Jan. 2013. DOI 10.1007/s11390-013-1319-3

A Secure Scalar Product Protocol Against Malicious Adversaries

Bo Yang[1] (杨　波), Yong Yu[2] (禹　勇), and Chung-Huang Yang[3] (杨中皇)

[1] *School of Computer Science, Shaanxi Normal University, Xi'an 710062, China*

[2] *School of Computer Science and Engineering, University of Electronic Science and Technology of China Chengdu 610054, China*

[3] *Graduate Institute of Information and Computer Education, National Kaohsiung Normal University, Taiwan, China*

E-mail: byang@snnu.edu.cn; yuyong@uestc.edu.cn; chyang@nknucc.nknu.edu.tw

Received October 11, 2011; revised April 5, 2012.

Abstract A secure scalar product protocol is a type of specific secure multi-party computation problem. Using this kind of protocol, two involved parties are able to jointly compute the scalar product of their private vectors, but no party will reveal any information about his/her private vector to another one. The secure scalar product protocol is of great importance in many privacy-preserving applications such as privacy-preserving data mining, privacy-preserving cooperative statistical analysis, and privacy-preserving geometry computation. In this paper, we give an efficient and secure scalar product protocol in the presence of malicious adversaries based on two important tools: the proof of knowledge of a discrete logarithm and the verifiable encryption. The security of the new protocol is proved under the standard simulation-based definitions. Compared with the existing schemes, our scheme offers higher efficiency because of avoiding inefficient cut-and-choose proofs.

Keywords secure multi-party computation, secure scalar product protocol, verifiable encryption

1 Introduction

A secure multi-party computation (SMC) is a type of protocol where two or more involved parties compute a commonly agreed function, with the restriction that none of the participants can learn anything more than their own inputs and the public outputs. A secure scalar product protocol is a specific SMC problem, and its goal is that two parties jointly compute the scalar product of their private vectors and no party will reveal any information about his/her private vector to another party. As a building block, the secure scalar product protocol has found broad applications in many areas such as privacy-preserving data mining[1-3], privacy-preserving cooperative statistical analysis[4], privacy-preserving geometry computation[5-8].

The secure scalar product protocol deals with the following problem: Let Alice has a private vector $\boldsymbol{X} = (x_1, \ldots, x_N)$ and Bob has another private vector $\boldsymbol{Y} = (y_1, \ldots, y_N)$. They compute corporately $u = \boldsymbol{X} \cdot \boldsymbol{Y} + v = \sum_{i=1}^{N} x_i y_i + v$, where u is a uniformly distributed random number known by Alice and v is a dependent uniformly distributed random number known by Bob. The purpose of Bob's random v is to prevent Alice from knowing the final result of $\boldsymbol{X} \cdot \boldsymbol{Y}$, and to ensure the fairness for Bob.

Many efforts to solving this problem have been done [2-5,9-10] and Goethals *et al.* gave a state-of-the-art overview of the problem and the properties of some solutions in [2].

Goethals *et al.*[2] demonstrated that if a vector has a low support (the support of a vector is defined by the number of non-zero elements in this vector), another party will learn half elements of this vector and further learn the whole vector with a high probability. Then they proposed a new secure scalar product protocol based on homomorphic encryption with plaintext space Z_m for some large m. The protocol is secure in the semi-honest model, assuming that $\boldsymbol{X}, \boldsymbol{Y} \in Z_\mu^N$, in which $\mu = \lfloor \sqrt{m/N} \rfloor$. However, all of the protocols mentioned above assume participants to be semi-honest.

Artak[10] gave a secure scalar product protocol based on homomorphic encryption and permutation on vectors. However, the protocol only considers a special case $v = 0$, in which the first party is able to get the final result of $u = \boldsymbol{X} \cdot \boldsymbol{Y}$, which is unfair for the second party.

All the other protocols in [3-5, 9] also consider participants to be semi-honest.

Short Paper

This work was supported by the National Natural Science Foundation of China under Grant Nos. 60973134, 61173164, 61003232, and the Natural Science Foundation of Guangdong Province of China under Grant No. 10351806001000000.

Hazay[11] presented an oblivious polynomial evaluation protocol (OPE) against malicious adversaries, where there are two parties, Alice and Bob, and Alice has a polynomial $p()$ and Bob has an input x. They would like to collaborate in a way for Bob to compute $p(x)$ such that Alice learns nothing about x and Bob learns only $p(x)$ and nothing more. The author claimed the protocol can be modified to compute the secure scalar product functionality. However, the performance of the scheme is not satisfying since inefficient cut-and-choose proofs are employed. In this paper, we will give a secure scalar product protocol against malicious adversaries by incorporating proof of knowledge and the verifiable encryption, which is more efficient in performance.

2 Fundamental Concepts and Building Blocks

2.1 Concepts in Computational Complexity

Negligible Function. A function $\mu \mapsto (0,1)$ is called negligible if for every positive polynomial $p(x)$, and all sufficiently large n's, $\mu(n) < 1/p(n)$.

Probability Ensembles. A probability ensemble indexed by $S \subseteq \{0,1\}^*$ is a family $\{E_w\}_{w\in S}$, so that each E_w is a random variable (or distribution) which ranges over (a subset of) $\{0,1\}^{poly(|w|)}$. Typically, we consider $S = \{0,1\}^*$ and $S = \{1^n : n \text{ is a natural number}\}$.

Identically Distributed. We say that two ensembles, $E \overset{\text{def}}{=} \{E_w\}_{w\in S}$ and $F \overset{\text{def}}{=} \{F_w\}_{w\in S}$, are identically distributed, and write as $E \equiv F$, if for every $w \in S$ and every α

$$Pr(E_w = \alpha) = Pr(F_w = \alpha), \quad (1)$$

where Pr means probability.

Computationally Indistinguishable. Two ensembles, $E = \{E_w\}_{w\in S}$ and $F = \{F_w\}_{w\in S}$, are computationally indistinguishable, and denote as $X \overset{C}{\equiv} Y$, if for every non-uniform distinguisher D there exists a negligible function $\mu(\cdot)$ such that for every $a \in \{0,1\}^*$,

$$|Pr(D(E_w(a,n)) = 1) - Pr(D(F_w(a,n)) = 1)| < \mu(n). \quad (2)$$

2.2 Secure Computation

In this subsection we review the definition of the secure 2-party computation, which follows [12].

A 2-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). Let $f : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^* \times \{0,1\}^*, f = (f_1, f_2)$ be a functionality, where $f_1(x,y)$ (resp., $f_2(x,y)$) denotes the first (resp., second) element of $f(x,y)$, and Π be a 2-party protocol in which the first party (with input x) wishes to obtain $f_1(x,y)$ and the second party (with input y) wishes to obtain $f_2(x,y)$.

In a 2-party protocol, all parties can be classified into two categories, semi-honest and malicious. Roughly speaking, a semi-honest party is one who follows the protocol properly with the exception that it keeps a record of all its intermediate computations and might derive the other parties' inputs from the record. A malicious adversary may arbitrarily deviate from the specified protocol. When considering malicious adversaries, there are certain undesirable actions that cannot be prevented. Specifically, a party may refuse to participate in the protocol, substitute its local input (and enter with a different input), or abort the protocol prematurely. The security of a protocol is analyzed by comparing what an adversary can do in the protocol with what it can do in an ideal scenario that is secure. This is formalized by considering an ideal computation involving an incorruptible trusted third party to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns each party its respective output. Concretely, an execution in the ideal model proceeds as follows:

- *Inputs.* Each party obtains an input, denoted by u.
- *Sending inputs to trusted party.* An honest party always sends u to the trusted party. A malicious party may, depending on u (as well as on an auxiliary input and its coin tosses), either abort or send some $u' \in \{0,1\}^{|u|}$ to the trusted party.
- *The trusted party answers the first party.* After receiving an input pair (x,y), the trusted party computes $(f_1(x,y), f_2(x,y))$ and replies to the first party with $f_1(x,y)$. Otherwise, if it receives only one input, it replies to both parties with a special symbol, denoted by $\perp$.
- *The trusted party answers the second party.* If the first party is malicious, it may, depending on its input and the trusted party's answer, decide to stop the trusted party. In this case the trusted party sends $\perp$ to the second party. Otherwise, the trusted party sends $f_2(x,y)$ to the second party.
- *Output.* The honest party always outputs the message it has obtained from the trusted party. The malicious party may output an arbitrary (polynomial-time computable) function of its initial input and the message it has obtained from the trusted party.

A real model is a real (2-party) protocol, where there exits no trusted third party, and the malicious party may follow an arbitrary feasible strategy. In particular, the malicious party may abort the execution at any point in time, and when this happens prematurely, the other party is left with no output.

Having defined the ideal and real models, we can now define the security of the protocol. Loosely speaking, the definition asserts that a secure 2-party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that admissible pairs in the ideal model are able to simulate admissible pairs in an execution of a secure real-model protocol.

Formally, let $f = (f_1, f_2)$ be a probabilistic polynomial-time functionality and Π be a 2-party protocol for computing f. $\overline{A} = (A_1, A_2)$ be two parties in the real model (represented as non-uniform probabilistic expected polynomial-time machines). Such a pair is admissible if for at least one $i \in \{1, 2\}$, A_i is honest (i.e., follows the strategy specified by Π). Then, the joint execution of Π under $\overline{A}$ in the real model (on input pair (x, y)), denoted by $REAL_{\Pi,\overline{A}}$, is defined as the output pair of A_1 and A_2 resulting from the protocol interaction. Again, let $\overline{B} = (B_1, B_2)$ be two parties in the ideal model (represented as non-uniform probabilistic expected polynomial-time machines), which is also admissible. Then, the joint execution of Π under $\overline{B}$ in the ideal model (on input pair (x, y)), denoted by $IDEAL_{f,\overline{B}}$, is defined as the output pair of B_1 and B_2 from the ideal execution.

Definition 1 (Security in the Malicious Model). *Let f and Π be as above. Protocol Π is said to securely compute f (in the malicious model) if for every pair of admissible non-uniform probabilistic expected polynomial-time machines $\overline{A} = (A_1, A_2)$ for the real model, there exists a pair of admissible non-uniform probabilistic expected polynomial time machines $\overline{B} = (B_1, B_2)$ for the ideal model, such that*

$$\{IDEAL_{f,\overline{B}}(x, y)\}_{x,y \text{ s.t. } |x|=|y|} \stackrel{C}{\equiv} \{REAL_{\Pi,\overline{A}}(x, y)\}_{x,y \text{ s.t. } |x|=|y|}.$$

2.3 Building Blocks

In order to obtain the security against malicious adversaries, participants are required to prove the correctness of each protocol step. We use $PoK\{a|\Phi(a)\}$ to denote a zero-knowledge proof of value a that satisfies a publicly computable relation Φ. We will make use of the following proof system.

Proof of Knowledge of a Discrete Logarithm. Let two participants, Alice and Bob, know g and y, but only Alice has the knowledge of x such that $y = g^x$. This relation is denoted by

$$\mathcal{R}_{\rm DH} = \{(g, y, (x))|y = g^x\}.$$

The zero-knowledge proof of knowledge for $\mathcal{R}_{\rm DH}$, denoted by $\pi_{\rm DH}$, is given in [13].

Verifiable Encryption. A verifiable encryption[14] is a protocol that allows a prover to convince a verifier that a ciphertext encrypts a plaintext satisfying a certain binary relation $\mathcal{R} = M \times \Delta$.

For $\delta \in \Delta$, an element $m \in M$ such that $(m, \delta) \in \mathcal{R}$ is called a witness for δ. In the scheme, the encryptor is given a value δ, a witness m for δ, and then encrypts m to yield a ciphertext ψ and proves to another party that ψ can be decrypted to a witness for δ.

The discrete logarithm relation $\mathcal{R}$ is defined as follows.

Definition 2. *Let Γ be a cyclic group of order ρ generated by γ, assume that γ and ρ are publicly known, and that ρ is prime. Let $M = [\rho], \Delta = \Gamma$. The discrete logarithm relation is $\mathcal{R} = \{(m, \delta) \in M \times \Delta : \gamma^m = \delta\}$.*

The proof of a discrete logarithm relation means, given γ, δ and a ciphertext ψ, the encryptor is able to prove to the decryptor that ψ is an encryption $\log_\gamma \delta$ under some public key *pk*.

In [14], a practical verifiable encryption protocol for discrete logarithms in conjunction with an encryption algorithm, depicted in the following, is given.

Encryption Algorithm.

- *Setup.* Let $n = pq$ be the RSA-modulo with $p = 2p' + 1, q = 2q' + 1$, where p, q, p', q' are primes. Choose $g' \in_R Z^*_{n^2}$, compute $g = (g')^{2n} \pmod{n^2}$, and publish (n, g) as system parameters.
- *Key Generation.* Choose a random $x \in_R [n^2/4]$ as the secret key, and compute $y = g^x \pmod{n^2}$ as the public key, where $[a]$ denotes the set $\{0, \ldots, \lfloor a-1 \rfloor\}$ for real number a.
- *Encryption.* For $m \in [n]$, choose a random $r \in_R [n/4]$ and compute

$$u = g^r \pmod{n^2}, \quad e = y^r h^m \pmod{n^2},$$

where $h = (1 + n) \pmod{n^2} \in Z^*_{n^2}$.

The ciphertext is (u, e).

- *Decryption.* For a ciphertext $c = (u, e) \in Z^*_{n^2} \times Z^*_{n^2}$, let $t = 2^{-1} \pmod{n}$, and compute $\widehat{m} = (e/u^x)^{2t}$. If $\widehat{m}$ is of the form h^m for some $m \in [n]$, then output m; otherwise, output *reject*.

The algorithm is semantic secure under the Composite Residuesity Assumption[15] on $Z^*_{n^2}$.

In the following, we will denote the encryption process by $E_{pk}(m)$, or $E_{pk}(m; r)$ when no misunderstanding is possible, and the decryption process by $D_{sk}(c)$.

The scheme is additive homomorphic since

$$\begin{aligned} E_{pk}(m_1; r_1) \odot E_{pk}(m_2; r_2) &= (g^{r_1}, y^{r_1} h^{m_1}) \odot (g^{r_2}, y^{r_2} h^{m_2}) \\ &= (g^{r_1+r_2}, y^{r_1+r_2} h^{m_1+m_2}) \\ &= E_{pk}(m_1 + m_2; r_1 + r_2). \end{aligned}$$

The verifiable encryption scheme was used in the construction of an oblivious pseudorandom function in [16]. In the following, we will use it to construct a secure scalar product protocol. For notational simplicity, we will denote the protocol of a verifiable encryption non-interactively by

$$PoK\{m|\psi \in Enc_{pk}(m) \wedge \delta = \gamma^m\},$$

where Enc_{pk} means an encrypton algorithm.

3 Secure Scalar Product Protocol

In the following, we build a secure scalar product protocol against malicious adversaries based on the solution in [2], where γ is the public base in the discrete logarithm relation.

3.1 Scheme Description

Protocol 1 (π_{sp}–Secure Scalar Product Protocol).

Inputs. A_1 has a private vector $\boldsymbol{X} = (x_1, \ldots, x_N)$ and A_2 has another private vector $\boldsymbol{Y} = (y_1, \ldots, y_N)$, $x_i, y_i \in [n]$ for $i = 1, \ldots, N$.

Outputs. A_1 gets u, and A_2 gets v, satisfying $u, v \in [n]$ and $u = \sum_{i=1}^N x_i y_i + v$.

1) Key generation: A_1 chooses a random $sk \in_R [n^2/4]$ as the secret key, and computes $pk = g^{sk}$ as the public key. A_1 sends pk to A_2, and engages in a zero-knowledge proof of knowledge $\pi_{\rm DH}$ with A_2, in which A_1 proves that $(g, pk, (sk)) \in \mathcal{R}_{\rm DH}$. This step is denoted by $(pk, sk) \leftarrow_R$ KGen.

2) A_1 computes $(c_i, \delta_i^{(A_1)}) = (E_{pk}(x_i), \gamma^{x_i})$, $\pi_i^{(A_1)} = PoK\{x_i | c_i = E_{pk}(x_i) \wedge \delta_i^{(A_1)} = \gamma^{x_i}\}$, and sends $\{(c_i, \delta_i^{(A_1)}), \pi_i^{A_1}\}$ $(i = 1, \ldots, N)$ to A_2.

3) A_2 verifies $\pi_i^{(A_1)}$ $(i = 1, \ldots, N)$. If there exists an $i \in \{1, \ldots, N\}$ such that the verification of $\pi_i^{(A_1)}$ fails, A_2 aborts with output $\perp$. Otherwise, A_2 chooses $v_i \in [n]$ and computes $(C_i, \delta_i^{(A_2)}) = (c_i^{y_i} E_{pk}(v_i), \gamma^{v_i}(\delta_i^{(A_1)})^{y_i})$, $\pi_i^{(A_2)} = PoK\{y_i | c_i^{y_i} E_{pk}(v_i) \wedge \gamma^{v_i}(\delta_i^{(A_1)})^{y_i}\}$, and sends $\{(C_i, \delta_i^{(A_2)}), \pi_i^{A_2}\}$ $(i = 1, \ldots, N)$ to A_1.

4) A_1 verifies $\pi_i^{(A_2)}$ $(i = 1, \ldots, N)$. If there exists an $i \in \{1, \ldots, N\}$ such that the verification of $\pi_i^{(A_2)}$ fails, A_1 aborts with output $\perp$. Otherwise, A_1 computes $u = D_{sk}(\prod_{i=1}^N C_i)$.

5) A_1 outputs u, and A_2 outputs $v = \sum_{i=1}^N v_i$.

In step 3,

$$\begin{aligned}(C_i, \delta_i^{(A_2)}) &= (c_i^{y_i} E_{pk}(v_i), \gamma^{v_i}(\delta_i^{(A_1)})^{y_i}) \\ &= (E_{pk}(x_i y_i + v_i), \gamma^{x_i y_i + v_i}).\end{aligned}$$

It is straightforward to verify that

$$u = \textstyle\sum_{i=1}^N x_i y_i + v.$$

3.2 Security

Theorem 1. *In the protocol π_{sp}, assume that the encryption scheme is additively homomorphic semantically secure, and each proof system is zero-knowledge and simulation-sound, the protocol π_{sp} securely computes the scalar product functionality, denoted by $\mathcal{F}_{sp}$, in the presence of malicious adversaries.*

Proof. To prove the theorem, we need to construct parties $\overline{B} = (B_1, B_2)$ in the ideal model from parties $\overline{A} = (A_1, A_2)$ in the real-model such that

$$\{\text{IDEAL}_{\mathcal{F}_{sp}, \overline{B}}(x, y)\} \stackrel{C}{\equiv} \{\text{REAL}_{\pi_{sp}, \overline{A}}(x, y)\}. \qquad (3)$$

We will consider the case that A_1 is malicious and the case that A_2 is malicious separately.

In the case that A_1 is malicious, because A_2 is honest, B_2 is taken as the same with A_2. We need to construct B_1 from a malicious real model A_1. The construction is made by having B_1 emulate an execution of A_1, while playing A_2's role. B_1 does this when interacting with the trusted third party, denoted by $\mathcal{F}_{sp}$, and its aim is to obtain an execution with A_1 that is consistent with the output received from $\mathcal{F}_{sp}$. B_1 works as follows.

1) B_1 receives from A_1 a public key pk and plays the verifier in $\pi_{\rm DH}$ with A_1 as the prover. If it does not accept the proof, it sends $\perp$ to the trusted third party and outputs whatever A_1 outputs. If it accepts the proof, then it runs the knowledge extractor for $\pi_{\rm DH}$ in order to obtain the witness sk used by A_1. If B_1 does not succeed in extracting, it outputs fail.

2) B_1 receives $(c_i, \delta_i^{(A_1)}, \pi_i^{(A_1)})$ from A_1, and verifies $\pi_i^{(A_1)}$ $(i = 1, \ldots, N)$. If there exists an $i \in \{1, \ldots, N\}$ such that the verification of $\pi_i^{(A_1)}$ fails, B_1 aborts with output $\perp$. Otherwise, it decrypts c_i $(i = 1, \ldots, N)$ to obtain $x_1, \ldots, x_N$.

3) B_1 picks $y_1'', \ldots, y_N'', v_1'', \ldots, v_N'' \in_R [n]$, sends $(\overline{C_i} = E_{pk}(y_i''), \overline{\delta}_i^{(B_1)} = \gamma^{v_i''})$ to A_1, and simulates the proof $\pi_i^{(A_2)}$ $(i = 1, \ldots, N)$.

4) If all $\pi_i^{(A_2)}$ $(i = 1, \ldots, N)$ pass the verification, B_1 sends $x_1, \ldots, x_N$ to $\mathcal{F}_{sp}$. $\mathcal{F}_{sp}$ receives $y_1, \ldots, y_N$ and v from the the ideal model B_2, computes $u = \sum_{i=1}^N x_i y_i + v$, and sends u to A_1.

5) B_1 outputs whatever A_1 does.

We need to show that (3) can be satisfied.

This can be proven by constructing a series of games $Game_l$, in which $B_1^{(l)}$ is a simulator constructed in the l-th hybrid game, $H_{B_1^{(l)}(z), B_2}(x, y)$ is the joint output distribution of $B_1^{(l)}$ and B_2, z is the auxiliary input of $B_1^{(l)}$, the last game $Game_4$ runs the protocol among B_1, B_2 and $\mathcal{F}_{sp}$.

Game$_1$. In the first game, we define a simulator $B_1^{(1)}$ who works exactly like A_2 in the protocol π_{sp} except that instead of acting as the verifier in the proof of $\pi_{\rm DH}$ in step 1, $B_1^{(1)}$ runs the extractor algorithm for $\mathcal{R}_{\rm DH}$ with A_1 to extract sk. $B_1^{(1)}$ outputs A_1's output.

We have $\{H_{B_1^{(1)}(z),B_2}(x,y)\} \equiv \{\text{REAL}_{\pi_{sp},\overline{A}}(x,y)\}$.

Game$_2$. In this game, we define a simulator $B_1^{(2)}$ who works exactly like $B_1^{(1)}$ in *Game*$_1$ except that instead of acting as the verifier in the proof of $\pi_i^{(A_1)}$ in step 2, $B_1^{(2)}$ decrypts c_i to obtain x_i ($i = 1,\ldots,N$). $B_1^{(2)}$ outputs $B_1^{(1)}$'s output.

We have $\{H_{B_1^{(2)}(z),B_2}(x,y)\} \equiv \{H_{B_1^{(1)}(z),B_2}(x,y)\}$.

Game$_3$. In this game, we define a simulator $B_1^{(3)}$ who works exactly like $B_1^{(2)}$ in *Game*$_2$ except that in step 3, $B_1^{(3)}$ picks $y'_1,\ldots,y'_N,v'_1,\ldots,v'_N \in_R [n]$, computes $(\overline{C_i},\overline{\delta}_i^{(B_1)}) = (c_i^{y'_i}E_{pk}(v'_i),\gamma^{v'_i}(\delta_i^{(A_1)})^{y'_i})$, $\overline{\pi_i}^{(A_2)} = PoK\{y'_i|c_i^{y'_i}E_{pk}(v'_i) \wedge \gamma^{v'_i}(\delta_i^{(A_1)})^{y'_i}\}$, and sends $((\overline{C_i},\overline{\delta}_i^{(B_1)}),\overline{\pi_i}^{(A_2)})$ to A_1.

By the indistinguishability of encryption, we have

$$\{H_{B_1^{(3)}(z),B_2}(x,y)\} \stackrel{C}{\equiv} \{H_{B_1^{(2)}(z),B_2}(x,y)\}.$$

Game$_4$: In this game, we define a simulator $B_1^{(4)}$ who works exactly like $B_1^{(3)}$ in *Game*$_3$ except that instead of proving $\overline{\pi_i}^{(A_2)}$, $B_1^{(4)}$ simulates the proof $\overline{\pi_i}^{(A_2)}$.

By zero-knowledge of $\pi_i^{(A_2)}$ ($i = 1,\ldots,N$), we have

$$\{H_{B_1^{(4)}(z),B_2}(x,y)\} \stackrel{C}{\equiv} \{H_{B_1^{(3)}(z),B_2}(x,y)\}.$$

Game$_5$: In this game, we define a simulator $B_1^{(5)}$ who works exactly like $B_1^{(4)}$ in *Game*$_4$ except that instead of computing $(\overline{C_i},\overline{\delta}_i^{(B_1)})$, $B_1^{(5)}$ picks $y''_1,\ldots,y''_N,v''_1,\ldots,v''_N \in_R [n]$, sends $(\overline{C_i} = E_{pk}(y''_i),\overline{\delta}_i^{(B_1)} = \gamma^{v''_i})$ to A_1, and simulates the proof $\overline{\pi_i}^{(A_2)}$ ($i = 1,\ldots,N$).

By the indistinguishability of encryption, we have

$$\{H_{B_1^{(5)}(z),B_2}(x,y)\} \stackrel{C}{\equiv} \{H_{B_1^{(4)}(z),B_2}(x,y)\}.$$

Game$_6$. *Game*$_6$ is the ideal model game among B_1 (with access to A_1), $\mathcal{F}_{sp}$ and B_2. Instead of computing $u = D_{sk}(\prod_{i=1}^N C_i)$ by A_1 as the last step of *Game*$_5$, B_1 sends $(x_1,\ldots,x_N)$ to $\mathcal{F}_{sp}$. $\mathcal{F}_{sp}$ computes $u = \sum_{i=1}^N x_iy_i + \sum_{i=1}^N v_i$ on $(x_1,\ldots,x_N)$ given by B_1 and $(y_1,\ldots,y_N,v_1,\ldots,v_N)$ given by B_2, and sends u to A_1.

It is obvious that

$$\{\text{IDEAL}_{\mathcal{F}_{sp},\overline{B}}(x,y)\} \equiv \{H_{B_1^{(5)}(z),B_2}(x,y)\}.$$

From *Game*$_1$ to *Game*$_6$, two joint output distributions $\{H_{B_1^{(l-1)}(z),B_2}(x,y)\}$ and $\{H_{B_1^{(l)}(z),B_2}(x,y)\}$, $l = 2,\ldots,6$, are either identical or computationally indistinguishable. So we have

$$\{\text{IDEAL}_{\mathcal{F}_{sp},\overline{B}}(x,y)\} \stackrel{C}{\equiv} \{\text{REAL}_{\pi_{sp},\overline{A}}(x,y)\}.$$

In the case that A_2 is malicious, because A_1 is honest, B_1 is viewed as the same with A_1. We need to construct B_2 from a malicious real model A_2. The construction is made by having B_2 emulate an execution of A_2, while playing A_1's role. B_2 works as follows.

1) B_2 runs $(pk',sk') \leftarrow_R$ KGen, sends pk' to A_2, and simulates the proof $\pi_{\rm DH}$.

2) B_2 picks $x'_i \in_R [n]$, sets $(\overline{c_i},\overline{\delta}_i^{(A_1)}) = (E_{pk'}(x'_i),\gamma^{x'_i})$, sends $(\overline{c_i},\overline{\delta}_i^{(A_1)})$ to A_2 and simulates the proof $\pi_i^{(A_1)}$ for $i = 1,\ldots,N$.

3) If all proofs $\pi_i^{(A_1)}$ pass the verification, A_2 sends $\{(C_i,\delta_i^{(A_2)}),\pi_i^{(A_2)}\}$ ($i = 1,\ldots,N$) to B_2.

4) B_2 verifies $\pi_i^{(A_2)}$ ($i = 1,\ldots,N$). If there exists an $i \in \{1,\ldots,N\}$ such that the verification of $\pi_i^{(A_2)}$ fails, B_2 aborts with output $\perp$. Otherwise, B_2 extracts $y_1,\ldots,y_N$, $v_1,\ldots,v_N$, and sends them to $\mathcal{F}_{sp}$.

5) $\mathcal{F}_{sp}$ computes $u = \sum_{i=1}^N x_iy_i + \sum_{i=1}^N v_i$ on B_2's input and B_1's input $x_1,\ldots,x_N$, and outputs u to B_1.

6) B_2 outputs whatever A_2 does.

Below, we show that

$$\{\text{IDEAL}_{\mathcal{F}_{sp},\overline{B}}(x,y)\} \stackrel{C}{\equiv} \{\text{REAL}_{\pi_{sp},\overline{A}}(x,y)\}.$$

Similarly, we construct a series of games where $B_2^{(l)}$ is a simulator constructed in the l-th hybrid game, $H_{B_1,B_2^{(l)}(z)}(x,y)$ is the joint output distribution of B_1 and $B_2^{(l)}$, and z is the auxiliary input of $B_2^{(l)}$.

Game$_1$. In the first game, we define a simulator $B_2^{(1)}$ who works exactly like A_1 in the protocol π_{sp} except that instead of acting as the real prover in the proof of $\pi_{\rm DH}$ in step 1, $B_2^{(1)}$ picks $(pk',sk') \leftarrow_R$ KGen, sends pk' to A_2, and simulates the proof $\pi_{\rm DH}$. $B_2^{(1)}$ outputs A_1's output.

We have $\{H_{B_1,B_2^{(1)}(z)}(x,y)\} \equiv \{\text{REAL}_{\pi_{sp},\overline{A}}(x,y)\}$.

Game$_2$. In this game, we define a simulator $B_2^{(2)}$ who works exactly like $B_2^{(1)}$ in *Game*$_1$ except that instead of acting as the real prover in the proof of $\pi_i^{(A_1)}$ ($i = 1,\ldots,N$) in step 2, $B_2^{(2)}$ picks $x'_i \in_R [n]$, computes $(\overline{c_i},\overline{\delta}_i^{(B_2^{(2)})}) = (E_{pk'}(x'_i),\gamma^{x'_i})$, $\overline{\pi_i}^{(B_2^{(2)})} = PoK\{x'_i|\overline{c_i} = E_{pk}(x'_i) \wedge \overline{\delta}_i^{(B_2^{(2)})} = \gamma^{x'_i}\}$, and sends $\{(\overline{c_i},\overline{\delta}_i^{(B_2^{(2)})}),\overline{\pi}_i^{(B_2^{(2)})}\}(i = 1,\ldots,N)$ to A_2. $B_2^{(2)}$ outputs $B_2^{(1)}$'s output.

By the indistinguishability of encryption, we have

$$\{H_{B_1,B_2^{(2)}(z)}(x,y)\} \stackrel{C}{\equiv} \{H_{B_1,B_2^{(1)}(z)}(x,y)\}.$$

Game$_3$. In this game, we define a simulator $B_2^{(3)}$ who works exactly like $B_2^{(2)}$ in *Game*$_2$ except that instead of proving $\overline{\pi}_i^{(B_2^{(2)})}$, it simulates the proof $\overline{\pi}_i^{(B_2^{(2)})}$ for $i=1,\ldots,N$.

By zero-knowledge of $\overline{\pi}_i^{(B_2^{(2)})}$ for $i=1,\ldots,N$, we have $\{H_{B_1,B_2^{(3)}(z)}(x,y)\} \stackrel{C}{\equiv} \{H_{B_1,B_2^{(2)}(z)}(x,y)\}$.

Game$_4$: In this game, we define a simulator $B_2^{(4)}$ who works exactly like $B_2^{(3)}$ in *Game*$_3$ except that instead of playing the verifier in the proof of $\pi_i^{(A_2)}$, in step 4, after receiving $\{(C_i,\delta_i^{(A_2)}),\pi_i^{(A_2)}\}$ $(i=1,\ldots,N)$, $B_2^{(4)}$ extracts $y_1,\ldots,y_N$, $v_1,\ldots,v_N$, and sends them to $\mathcal{F}_{sp}$. $\mathcal{F}_{sp}$ computes $u=\sum_{i=1}^{N}x_iy_i+\sum_{i=1}^{N}v_i$ on $x_1,\ldots,x_N$ given by B_1, and $y_1,\ldots,y_N$, $v_1,\ldots,v_N$ given by $B_2^{(4)}$, and sends u to B_1. In this game, $B_2^{(4)}$ works exactly like B in ideal model.

It is obvious that

$$\{H_{B_1,B_2^{(4)}(z)}(x,y)\} \equiv \{H_{B_1,B_2^{(3)}(z)}(x,y)\}.$$

So, in this case, also we have

$$\{\text{IDEAL}_{\mathcal{F}_{sp},\overline{B}}(x,y)\} \stackrel{C}{\equiv} \{\text{REAL}_{\pi_{sp},\overline{A}}(x,y)\}. \qquad \square$$

3.3 Resource Analysis

Computational Cost. In the verifiable encryption protocol, the modulo n in the encryption algorithm can be set $|n| = 1\,024$ bits, the order ρ of the cyclic group $\varGamma$ can be taken smaller than n. In the encryption algorithm, the encryptor and the decryptor require $3\log N$ modular multiplications $\pmod{n^2}$, respectively, where N is the dimension of vectors. In the proof of the binary relation $(m,\delta)\in\mathcal{R}$, the prover and the verifier require $3\log\rho$ modular multiplications $\pmod{\rho^2}$, respectively. In the proof of $\pi_{\rm DH}$, the prover requires $\log n$ modular multiplications $\pmod{n}$, the verifier requires $2\log n$ modular multiplications $\pmod{n}$. Therefore, in our protocol, two participants' computational costs, measured in the number of modular multiplications, are $6N\log n \pmod{n^2} + 6N\log\rho \pmod{\rho^2} + \log n \pmod{n}$ and $6N\log n \pmod{n^2} + 6N\log\rho \pmod{\rho^2} + 2\log n \pmod{n}$, respectively.

Communication Cost. In the encryption algorithm, $4\log n$ bits are exchanged, while in the proof of the binary relation $(m,\delta)\in\mathcal{R}$, $4\log\rho$ bits are exchanged. In the proof of $\pi_{\rm DH}$, $2\log n$ bits are exchanged, therefore, the communication cost is $8N\log n + 8N\log\rho + 2\log n$ bits.

In [2], the protocol, where the proof of knowledge and the verifiable encryption are not employed, was given in the semi-honest model, and cannot prevent the involved participants' malicious actions. Two participants' computational costs are both $6N\log n \pmod{n^2}$. The communication cost is $8N\log n$.

Similarly, in [10], the proof of knowledge and the verifiable encryption is not employed. The protocol was given in the semi-honest model, and cannot avoid the malicious actions of the involved participants. Furthermore, the protocol considers a special case $v=0$, in which the first party can get the final result $u=\boldsymbol{X}\cdot\boldsymbol{Y}$, thus, it is unfair for the second party.

The computational costs of two participants are $6N\log n \pmod{n^2} + 2N \pmod{n}$ and $3N\log n \pmod{n^2} + 2N \pmod{n}$, respectively. The communication cost is $8N\log n + 2$.

In [11], inefficient cut-and-choose proofs are used to prove that A_1 behaves correctly. A_2 sends a commitment to a challenge $\tau=\tau_1\ldots\tau_s$, where s is the statistical security parameter. A_1 then chooses s random polynomials $\{q_1,\ldots,q_s\}$ of degree N and sends all encryption of coefficients of q_i and $p-q_i$ $(i=1,\ldots,s)$ for checking A_1. So the cost (including computational cost and communication cost) is at least $2s$ times of that of ours.

We define $A = 3N\log n \pmod{n^2}, B = 6N\log\rho \pmod{\rho^2}, C=\log n \pmod{n}$ and $D=2N \pmod{n}$, the numbers of modular multiplications are ($\pmod{n^2}$, $\pmod{\rho^2}$, $\pmod{n}$ and $\pmod{n}$), respectively, and the numbers of bits exchanged between the involved participants are $E=8N\log n$, $F=8N\log\rho$, $G=2\log n$, respectively, and obtain the comparisons of our protocol with those in [2, 10] and in [11] about the computational costs (Comp.) and the communication costs (Comm.) in Table 1.

The first column is the names of the three protocols, the second column is the model of the involved participants, and the third and the fourth columns are

Table 1. Comparison of Four Protocols

Protocol	Model	Comp. of A_1	Comp. of A_2	Comm.
Ours	Malicious	$2A+B+C$	$2A+B+2C$	$E+F+G$
[2]	Semi-honest	$2A$	$2A$	E
[10]	Semi-honest	$2A+D$	$A+D$	$E+2$
[11]	Malicious	$2s(2A+B+C)$	$2s(2A+B+2C)$	$2s(E+F+G)$

the computational costs of the first party A_1 and the second party A_2 respectively, and the last column is the communication cost.

4 Conclusions

Based on two building blocks, namely the proof of knowledge of a discrete logarithm and the verifiable encryption, we have given a scalar product protocol in the presence of malicious adversaries. Under the standard simulation-based definitions, the protocol is proven secure. Compared with the existing schemes, our scheme offers higher efficiency because of avoiding inefficient cut-and-choose proofs.

References

[1] Tran D H, Ng W K, Lim H W *et al.* An efficient cacheable secure scalar product protocol for privacy-preserving data mining. In *Proc. the 13th Int. Conf. Data Warehousing and Knowledge Discovery*, Aug. 29-Sept. 2, 2011, pp.354-366.

[2] Goethals B, Laur S, Lipmaa H, Mielikainen T. On private scalar product computation for privacy-preserving data mining. In *Proc. the 7th Int. Conf. Information Security and Cryptology*, Dec. 2004, pp.104-120.

[3] Vaidya J, Clifton C. Privacy preserving association rule mining in vertically partitioned data. In *Proc. the 8th SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, July 2002, pp.639-644.

[4] Du W, Atallah M. Privacy-preserving cooperative statistical analysis. In *Proc. the 17th Annual Computer Security Applications Conference*, Dec. 2001, pp.102-110.

[5] Atallah M J, Du W. Secure multiparty computational geometry. In *Proc. the 7th International Workshop on Algorithms and Data Structures*, Aug. 2011, pp.165-179.

[6] Thomas T. Secure Two-party protocols for point inclusion problem. *Int. J. Network Security*, 2009, 9(1): 1-7.

[7] Yang B, Sun A D, Zhang W Z. Secure two-party protocols on planar circles. *Journal of Information & Computational Science*, 2011, 8(1): 29-40.

[8] Yang B, Shao Z Y, Zhang W Z. Secure two-party protocols on planar convex hulls. *Journal of Information & Computational Science*, 2012, 9(4): 915-929.

[9] Du W, Zhan Z. Building decision tree classifier on private data. In *Proc. IEEE ICDM Workshop on Privacy, Security, and Data Mining*, Dec. 2002, Vol.14, pp.1-8.

[10] Amirbekyan A, Estivill-Castro V E C. A new efficient privacy-preserving scalar product protocol. In *Proc. the 6th Australasian Data Mining Conference*, Dec. 2007, pp.209-214.

[11] Hazay C. Efficient two-party computation with simulation based security [Ph.D. Thesis]. Senate of Bar-Ilan University, Israel, 2009.

[12] Goldreich O. Foundations of Cryptography (Vol.2): Basic Applications. London, UK: Cambridge University Press, 2004.

[13] Schnorr C P. Efficient signature generation by smart cards. *Journal of Cryptology*, 1991, 4(3): 161-174.

[14] Camenisch J, Shoup V. Practical verifiable encryption and decryption of discrete logarithms. In *Proc. CRYPTO 2003*, Aug. 2003, pp.126-144.

[15] Paillier P. Public-key cryptosystems based on composite degree residue classes. In *Proc. the 17th Theory and Application of Cryptographic Techniques*, May 1999, pp.223-238.

[16] Jarecki S, Liu X. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *Proc. the 6th Theory of Cryptography Conference*, March 2009, pp.577-594.

Bo Yang received the B.S. degree from Peking University in 1986, and the M.S. and Ph.D. degrees from Xidian University in 1993 and 1999, respectively. From July 1986 to July 2005, he had been at Xidian University. From 2002, he had been a professor of National Key Lab. of ISN in Xidian University. He has served as a program chair for the fourth China Conference on Information and Communications Security in 2005, the vice-chair for ChinaCrypt 2009, and the general chair for the 5th Joint Workshop on Information Security. He is currently a professor and supervisor of Ph.D. candidates at the School of Computer Science, Shaanxi Normal University and a special-term professor of Shaanxi Province. His research interests include information theory and cryptography.

Yong Yu received the Ph.D. degree in cryptography from Xidian University in 2008. He is currently an associate professor at the School of Computer Science and Engineering, University of Electronic Science and Technology of China. His research focuses on cryptography and its applications, especially public encryption and digital signature.

Chung-Huang Yang received the B.S. degree from National Cheng-Kung University, Taiwan, China, in 1981 and the M.S. and Ph.D. degrees from University of Louisiana at Lafayette, USA, in 1986 and 1990, respectively. He is currently a professor at Graduate Institute of Information and Computer Education, National Kaohsiung Normal University, China, and a member of the Board of Executive Directors of (Taiwan) Chinese Cryptology and Information Security Association. His research interests include the implementation of cryptographic algorithms, computer forensics and mobile phone forensics.