Protecting User Privacy for Cloud Computing by Bivariate Polynomial Based Secret Sharing

Ching-Nung Yang¹, Jia-Bin Lai¹ and Zhangjie Fu²

¹Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan
²School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, China

Cloud computing is an Internet-based computing. In cloud computing, the service is fully served by the provider. Users need nothing but personal devices and Internet access. Computing services, such as data, storage, software, computing, and application, can be delivered to local devices through Internet. The major security issue of cloud computing is that cloud providers must ensure that their infrastructure is secure, and prevent illegal data accesses from outsiders, other clients, or even the unauthorized cloud employees. In this paper, we deal with key agreement and authentication for cloud computing. By using Elliptic Curve Diffie Hellman (ECDH) and symmetric bivariate polynomial based secret sharing, we design a secure cloud computing (SCC). Two types of SCC are proposed. One requires a trusted third party (TTP), and the other does not need a TTP. Additionally, via the homomorphism property of polynomial based secret sharing, our SCC can be extended to multi-server SCC (MSCC) to fit an environment where a multi-server system contains multiple servers to collaborate for serving applications.

Keywords: cloud computing, authentication, secret sharing; key agreement, symmetric bivariate polynomial, homomorphism

1. Introduction

Cloud computing is a type of Internet-based computing, and it is one of the foundations of the next generation of computing. Computing services, such as data storage, software, computing, and application, are delivered to local devices through Internet [1, 2]. In cloud computing, the service is fully served by the provider. Clients need nothing but personal devices and Internet access. The cloud computing can either be hosted on-site by the company or off-site such as Microsoft’s SkyDrive, Google Drive, Samsung’s S-Cloud service, Apple’s iCloud, Amazon’s Cloud Drive. Recent applications, e.g., multimedia streaming, virtual reality, and robotics [3, 4, 5], have used cloud computing to provide the services. Additionally, platforms like Google Apps (e.g., Gmail, Google Groups, Google Calendar, etc.), YouTube, Vimeo, Flickr, Slideshare and Skype, have adopted the cloud computing technology. As cloud computing becomes more and more popular, how to secure cloud computing and protect data security deserves studying. Some security issues in cloud computing are surveyed and studied in [6–16].

For providing cloud services, the sensitive data for all clients should be stored in the cloud host. At this time, the data security and the personal privacy are assured. The cloud provider guarantees these data and personal information in host database against all accesses of the unauthorized insiders or the malicious outsiders. Accordingly, some secure cloud computing schemes based on secret sharing approach were proposed [17–20]. For example, Yeh’s PASS (Privacy by Authentication and Secret Sharing) prevents client’s data privacy from the unauthorized access [17]. PASS adopts public key cryptosystem to encrypt its share. So, in Yeh’s PASS,
the client cannot recover the secret key. Encryption/decryption in PASS should be accomplished with the help of encryption server and query server. If we easily store the secret key in the client side, the secret key will be leaked out when the client’s device is compromised.

In this paper, we deal with cloud security services including key agreement and authentication described in [17], and solve the above weaknesses of PASS. By using symmetric bivariate polynomial based secret sharing, we design the secure cloud computing (SCC). Two types of SCC are proposed. One requires a trusted third party (TTP) in cloud like the scheme in [7], and the other does not need a TTP. Meantime, our SCC provides mutual authentication to avoid the other party does not need a TTP. Meantime, our SCC are proposed. One requires a trusted third party to fit a multi-server environment.

The paper is organized as follows. Section 2 gives some preliminaries. Two types of SCC are introduced in Section 3, and the MSCC is proposed in Section 4. Performance evaluation, security analysis, and applications are shown in Section 5, and Section 6 is conclusion.

2. Preliminary

2.1. Elliptic Curve Diffie Hellman Protocol and Elliptic Curve Cryptography

Elliptic Curve Diffie Hellman (ECDH) key agreement protocol is based on elliptic curve discrete logarithm problem. On the same security level, the computation by elliptic curve discrete logarithm is faster than the multiplicative group discrete logarithm, and thus has the computational advantage. In ECDH, Alice randomly selects $n_A$ and computes $P_A = n_A \times G$, where $G$ is a point on elliptic curve. Then, Alice sends $P_A$ to Bob through public channel. By the same approach, Bob sends $P_B = n_B \times G$ to Alice. Finally Alice and Bob can share a same secret key by calculating $K = n_A \times P_B = n_B \times P_A$.

On the other hand, Elliptic Curve Cryptography (ECC) can implement encryption and decryption. Alice and Bob choose the public/private key $(P_A, n_A)$ and $(P_B, n_B)$, respectively. Alice encrypts the message $m$ by using Bob’s public key $P_B$ and a random number $k$ as $C = \{k \times G, m + k \times P_B\}$. After receiving $C$, Bob decrypts this ciphertext by subtracting $(n_B \times k \times G)$ from $(M + k \times P_B)$ to decrypt the message $m$.

2.2. $(k,n)$ Secret Sharing

Secret sharing is one of main research topics in modern cryptography and it has been studied extensively in literature. In 1979, Shamir [22] and Blakley [23] independently proposed secret sharing solutions for safeguarding cryptographic keys. In a $(k,n)$ secret sharing, the dealer divides the secret into $n$ shares and distributes shares among $n$ shareholders in such a way that any $k$ or more than $k$ shares can reconstruct this secret; but any $(k - 1)$ or fewer than $(k - 1)$ shares cannot obtain any information of the secret.

In this paper, we use Shamir’s $(k,n)$ secret sharing in a bivariate polynomial type to implement our SCC and MSCC. Thus, we describe Shamir’s $(k,n)$ secret sharing (univariate polynomial based secret sharing) and the notion of bivariate polynomial based secret sharing.

2.2.1. $(k,n)$ Secret Sharing Based on Univariate Polynomial

A polynomial based $(k, n)$ secret sharing scheme was firstly proposed by Shamir [22]. By taking the secret data as $g_0$ (constant term) in the following $(k - 1)$-degree polynomial $g(x)$ where $p$ is a prime number and $g_i$ is integer in $GF(p)$, the dealer could construct $n$ shares $(x_i, g(x_i))$ by choosing $n$ different $x_i$, $i \in [1, n]$, and sends them to shareholders. 

$$g(x) = (g_0 + g_1 x + \ldots + g_{k-1} x^{k-1}) \mod p \quad (1)$$

In secret reconstruction, any $k$ shares (say $k$ shares $(x_1, g(x_1)), (x_2, g(x_2)), \ldots, (x_k, g(x_k))$) are used for reconstructing $g(x)$ via Lagrange interpolation formula in (2), and the secret is obtained from $g_0 = g(0)$.

$$g(x)=g(x_1)=\frac{(x-x_2)(x-x_3)\ldots(x-x_k)}{(x_1-x_2)(x_1-x_3)\ldots(x_1-x_k)}+\ldots+\frac{g(x_2)}{(x_2-x_1)(x_2-x_3)\ldots(x_2-x_k)}\frac{(x-x_1)(x-x_3)\ldots(x-x_k)}{(x_1-x_1)(x_1-x_2)\ldots(x_1-x_k)}+\ldots+\frac{g(x_k)}{(x_k-x_1)(x_k-x_2)\ldots(x_k-x_k)} \mod p \quad (2)$$
The univariate polynomial based secret sharing schemes can be used for cloud computing which many users share and distribute their data to servers. For example, in [24], the authors adopted Shamir’s $(k, n)$ secret sharing to design a new secret sharing that can reduce the amount of shares in cloud computing environment. The system in [24] has four entities, users, key-servers, data-servers, and a dealer. Users have multiple secrets (say $m$ pieces secrets) and distribute these secrets in the cloud system. Key-server keeps only an initial and uses a keyed pseudo-random number generator to generate all the shares, and meanwhile the data-servers store all the required shares for these $m$ pieces secrets. Actually, in this cloud system, the key-servers are composed of small-capacity servers (because they store only the initial) but the data-servers are composed of large-capacity servers (they store all the shares for $m$ pieces secrets). By subdividing servers into key-server and data-servers, this secret sharing scheme can reduce the amount of shares stored in servers.

2.2.2. $(k,n)$ Secret Sharing Based on Bivariate Polynomial

There are two types of secret sharing using bivariate polynomial. Some of them [25, 26, 27] use an asymmetric bivariate polynomial, and some [28, 29, 30] use a symmetric bivariate polynomial. All polynomials in [25–30] are of degree $(k − 1)$ for both variables. In [25, 26, 27], the dealer selects an asymmetric $(k−1)$-degree bivariate polynomial $F(x, y)$, and sends two univariate polynomials $g_i(x) = F(x, i)$ and $f_i(y) = F(i, y)$ to each shareholder $P_i$. On the other hand, in [28, 29, 30], the dealer uses a bivariate polynomial $F(x, y)$, with the symmetric property $F(x, y) = F(y, x)$, and sends only one univariate polynomial $f_i(y)$ to the shareholder $P_i$. When using symmetric bivariate polynomial, both polynomials $g_i(x)$ and $f_i(y)$ have the same coefficients (the symmetric property). Notice that this is why the dealer sends only one polynomial $f_i(y)$ to $P_i$. For most bivariate polynomial based schemes, using bivariate polynomial is to adopt the advantageous features $f_i(j) = g_j(i)$ when using the asymmetric polynomial, and $f_i(j) = f_j(i)$ when using the symmetric polynomial, respectively.

A $(k−1)$-degree bivariate polynomial in (3) is symmetric, where $F(x, y) = F(y, x)$. Obviously, we can select $a_{ij} = a_{ji}$ to construct this symmetric bivariate polynomial because

$$\sum_{0 \leq i, j \leq (k−1)} a_{ij}x^iy^j = \sum_{0 \leq i, j \leq (k−1)} a_{ji}y^ix^j.$$  

$$F(x, y) = \sum_{0 \leq i, j \leq k−1} a_{ij}x^iy^j \quad (3)$$

Same as Shamir’s secret sharing [22], we embed the secret in $a_{00} = F(0, 0)$. The dealer selects a symmetric bivariate polynomial, and sends a $(k−1)$-degree univariate polynomial $f(ID_i, y) = b_{00} + b_{10}y + \ldots + b_{(k−1)0}y^{k−1}$, $1 \leq i \leq n$, to the shareholder $P_i$, where $ID_i$ is his/her identification. This bivariate polynomial based secret sharing also has the threshold value $k$, and this can be proven by Vandermonde matrix. When $k$ shareholders collaborate, they reconstruct the polynomial $F(x, y)$ from $k$ shares. Then, the secret can be determined from $a_{00} = F(0, 0)$. In this paper, we use the symmetric bivariate polynomial based secret sharing to adopt its symmetric property to design SCC and MSCC.

2.3. Yeh’s PASS

By secret sharing approach, Yeh’s PASS [17] recovers the secret key from the received share from client and the share stored in Authentication Server (AS). After deriving the secret key, AS authenticate the client by verifying the hash value of this key. Meantime, Encryption Server (ES) encrypts the client’s data based on this secret key. If the authentication succeeds, AS forwards the query along with the secret key, to the Query Server (QS). Finally, QS returns either encrypted query or non-encrypted results to the client. Encryption/decryption in PASS should be accomplished with the help of ES and QS. If we easily store the secret key in the client side, the secret key will be leaked out when the client’s device is compromised.

The above weakness comes from that AS does not send the share to the client (note: only the client in Yeh’s PASS uses the public key of cloud server to encrypt its share and sends it to the server). This weakness of Yeh’s PASS cannot be solved by simply sending the shares to the client by AS using public key cryptosystem, because knowing public keys of all clients by AS
is hard to achieve but all clients know the public key of server is reasonable.

Our SCC avoids using public key cryptosystem to send the shares, and achieves the mutual authentication between the server and the client. To demonstrate our improvements, we only briefly describe the following phases in Yeh’s PASS: (i) secret key agreement, (ii) secret shares agreement, and (iii) client authentication between AS and the client. One can refer to other phases between AS, ES, and QS in Figure 1 of [17].

**Secret Key Agreement:**

Each client and AP agree on a secret key $K_i$ and two secret shares $CS_i$ (known by the client) and $SS_i$ (known by AS). The shares $CS_i$ and $SS_i$ are used for deriving the secret key $K_i$. ECDH algorithm, which adopts $y^2 = x^3 + ax + b$ over a prime $p$, is used for this key agreement.

(Step 1) AS and the client $i$ choose a random number $r_s \in GF(p)$ and $r_i \in GF(p)$, respectively.

(Step 2) AS computes a point $R_s = r_s \times G$ and sends it to the client. Also, the client computes and sends a point $R_i = r_i \times G$ to AS.

(Step 3) AS and the client respectively compute a point $Q_i = r_s \times R_i = r_i \times R_s$.

(Step 4) AS and the client choose the $x$-coordinate of $Q_i$ as the shared secret key $K_i$.

**Secret Shares Agreement:**

After $K_i$ is determined, two secret shares $CS_i$ and $SS_i$ are then generated separately by the client $i$ and AS, respectively.

(Step 1) AS and the client $i$ computes a point $(Q_i + D_i)$, where $D_i$ is the public key of client $i$, and let $a$ be the $x$-coordinate of this point.

(Step 2) AS and the client can construct a same polynomial $f(x) = K_i + ax$ (note: this one-degree polynomial can be used in a $(2,2)$ secret sharing).

(Step 3) AS and the client $i$ randomly generate their secret shares as $SS_i = (x_1, f(x_1))$ and $SS_i = (x_2, f(x_2))$, respectively.

(Step 4) After the secret shares are chosen, AS and the client remove $Q_i$ and $f(x)$ from their storages.

(Step 5) In a later query, the client $i$ presents his share $CS_i$ to AS by using the public key of AS. AS uses $CS_i$ and its own share $SS_i$ to reconstruct the polynomial $f(x)$ and determine the key $K_i = f(0)$.

**Client Authentication:**

When receiving an authentication request from a client, AS starts client authentication procedure. After the successful authentication, the client can access his own data.

(Step 1) The client $i$ sends an authentication request to AS, $\text{ENC}_{D_2}(\text{client’s counter} \parallel CS_i)$, which is encrypted by AS’s public key $D_2$.

(Step 2) AS uses its own private key $d_s$ to decrypt the received request by $\text{DEC}_{d_s}(\text{ENC}_{D_2}(\text{client’s counter} \parallel CS_i))$, and then retrieves the client’s counter and the secret share $CS_i$.

(Step 3) AS uses the decrypted $CS_i$ and the stored $SS_i$ to recover the secret key $K_i$ using Shamir’s $(2,2)$ secret sharing, as described in (Step 5) of Secret Key Agreement.

(Step 4) AS computes the hash value $h(K_i)$, and verifies whether this hash value equals the stored hash value or not. Also, the client’s counter should be the same as the server’s counter. Suppose that all the above are true, then the client $i$ is successfully authenticated.

3. The Proposed SCC

Yeh’s PASS scheme chooses not to store the secret key in the cloud server, but recovers this secret key from the received share from client and the share stored in server. For such key recovery, the client should use the public key of cloud server to encrypt and send its share to the server. Actually, ECC encryption/decryption was used in [17]. To avoid using public key cryptosystem, and achieving the mutual authentication between the server and the client, we use the symmetric property of secret sharing.

The main security features for the proposed SCC are shown as follows: (1) Even though the cloud server and the local client device are compromised, the secret key cannot be obtained. (2) Malicious insiders in cloud server and outsiders cannot determine the secret key. (3)
Client does not need the complex public cryptosystem to send the share to cloud server, but only uses the symmetric encryption instead. (4) The mutual authentication between client and server is achieved. (5) SCC can be extended to MSCC, and the authentication and key recovery of MSCC can be efficiently accomplished.

Two types of SCC are proposed. One requires a trusted third party (TTP), and the other does not. Both types of SCC can provide the mutual authentication between the client and the cloud server. In the proposed SCC without TTP, ECDH is used to share the same secret key and the same bivariate polynomial. Besides, we do not use the public cryptosystem to send the secret shares. Subsequently, both types of SCC are described. Notations in these protocols are first defined in Table 1.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_c, n_s$</td>
<td>random numbers used in ECDH for client and server, respectively, where $n_c &lt; n$ and $n_s &lt; n$</td>
</tr>
<tr>
<td>$P_c, P_s$</td>
<td>Public values used in ECDH for client and server, respectively, where $P_c = n_c \times G$ and $P_s = n_s \times G$</td>
</tr>
<tr>
<td>$K_i$</td>
<td>secret key shared between Client $i$ and cloud server</td>
</tr>
<tr>
<td>$S_c, S_s$</td>
<td>shares for client and server, respectively</td>
</tr>
<tr>
<td>$ID_i$</td>
<td>ID of Client $i$</td>
</tr>
<tr>
<td>$h(\cdot)$</td>
<td>one-way hash function</td>
</tr>
<tr>
<td>$r$</td>
<td>intermediate key used in mutual authentication and secret key recovery</td>
</tr>
<tr>
<td>$E_k(\cdot)/D_k(\cdot)$</td>
<td>symmetric encryption/decryption with the secret key $K$</td>
</tr>
</tbody>
</table>

**3.1. Key Agreement Protocol without TTP**

There are three phases, key sharing phase, mutual authentication phase, and key recovery phase in the key agreement without TTP. In this protocol, we use ECDH to generate the bivariate polynomial, which is used in secret sharing to establish the key agreement.

**Key Sharing Phase:**

In this phase, client $ID_i$ and the cloud server generate their respective shares $S_c$ and $S_s$. After successfully generating the shares, they discard all sensitive information that could be used for compromising the secret. Also, both sides store the hash value of secret key $h(K_i)$ for mutual authentication. Figure 1 shows the key sharing phase, and all the details are described step by step as follows.

(Step 1) Client selects a random $n_c$ and calculates $P_c = n_c \times G$. Then, it sends it to the cloud server.

(Step 2) The cloud server selects a random $n_s$ and calculates $P_s = n_s \times G$, and sends it to the client.

(Steps 3 and 4) Client and the cloud server calculate a point $Q$ on elliptic curve by $n_c \times P_s$ and $n_s \times P_c$, respectively.

(Steps 5 and 6) Client and the cloud server, respectively, choose the one-degree bivariate polynomial $F(x, y) = a_{00} + a_{01}x + a_{10}y + a_{11}xy$, where coefficients are determined in Eq. (4). The value of $a_{00}$ is used as the secret key $K_i$. Also, we choose $a_{01} = a_{10}$ to get the symmetric $F(x, y)$.

\[
\begin{align*}
\{ a_{00} &= \text{x-coordinate of } Q, \\
 a_{01} &= a_{10} = \text{x-coordinate of } (Q + P_c), \\
 a_{11} &= \text{y-coordinate of } (Q + P_c)
\} \quad (4)
\]

(Step 7) The client selects a random $X_c$ and computes $F(X_c, y) = f_c(y)$. Then, client uses $S_c = (X_c, f_c(y))$ as his share.
(Step 8) The cloud server selects a random $X_s$ and computes $F(X_s, y) = f_s(y)$. Then, server uses $S_s = (X_s, f_s(y))$ as its share.

(Step 9) The client discards $Q$ and $F(x, y)$, and stores $S_c$ and $h(K_i)$.

(Step 10) The cloud server discards $Q$ and $F(x, y)$, and stores $ID_i$, $S_s$ and $h(K_i)$ in database.

**Mutual Authentication Phase:**

In Yeh’s PASS, the authentication is accomplished by verifying the hash value of secret key. Instead of reconstructing the polynomial, we use the symmetric property in bivariate polynomial to share an intermediate key $K_{int}$. Then the mutual authentication is finished by using this intermediate key. This mutual authentication can protect the privacy of client, and only allows the authenticated client to access his own data. Meantime, the client can assure that he connects the real server not a faked one. All steps are shown in Figure 2, and the details are briefly described below.

![Figure 2. Mutual authentication phase in the key agreement without TTP.](image)

(Step 1) The client sends $\{ID_i, X_c\}$ to the cloud server.

(Step 2) The cloud server uses $ID_i$ to find the corresponding $f_s(y)$ from database. Then, it determines the intermediate key $K_{int}$ from $h(f_s(X_c))$.

(Step 3) The cloud server sends $X_s$ to the client.

(Step 4) The client also obtains the intermediate key $K_{int}$ from $h(f_s(X_c))$. Because $f_s(X_c) = f_c(X_s)$ (the symmetric property of $F(x, y)$), the client and the cloud server can share the same intermediate key $K_{int}$.

(Step 5) The client selects a random $r$ and computes $E_{K_{int}}(r)$.

(Step 6) The client sends $\{r, E_{K_{int}}(r)\}$ to the cloud server.

(Step 7) The cloud server authenticates client by verifying $D_K(E_{K}(r)) = r$.

(Step 8) The cloud server sends back $E_{K}(r+1)$ to the client.

(Step 9) The client authenticates the cloud server by verifying $D_K(E_{K}(r+1)) = (r+1)$.

**Key Recovery Phase:**

In Yeh’s PASS, encryption and decryption are accomplished with the help of ES and QS. If we easily store the secret key in the client side, the client will risk losing his secret key when the local device is compromised. In the proposed SCC, we do not store the secret key $K_i$ in the client. In SCC, we use the intermediate key to share the shares $S_c$ and $S_s$. Finally, the secret key $K_i$ is recovered, respectively, by the client and the server. All steps are shown in Figure 3, and the details are briefly described below.

![Figure 3. Key recovery phase in the key agreement without TTP.](image)

(Step 1 – Step 4) Same as the steps in mutual authentication, client and server share the intermediate key $K_{int}$.

(Step 5) The client sends $E_{K_{int}}(S_c)$ to the cloud server.

(Step 6) The cloud server sends $E_{K_{int}}(S_s)$ to the cloud server.

(Step 7) The cloud server computes $F(x, y)$ from $S_c$ and $S_s$, and derives $F(0,0)$ to verify $h(F(0,0)) = h(K_i)$.
(Step 8) The client does the same operation as in (Step 7).

(Step 9) Server and client use $F(0, 0)$ as the secret key $K_i$, and then use $E_{K_i}()$ to secure their transmission.

(Step 10) Moreover, the server can use $E_{K_i}()$ to encrypt the stored data in the cloud server to protect the client’s privacy.

3.2. Key Agreement Protocol with TTP

Suppose that we have a TTP in SCC system model [7]. Then, the polynomial $F(x, y) = a_{00} + a_{01}x + a_{10}y + a_{11}xy$ can be generated by TTP. Afterwards, TTP sends $\{S_c, h(K_i)\}$ and $\{S_s, h(K_i)\}$ to the client and the cloud server, respectively, through a secure VPN channel. The mutual authentication phase and the key recovery phase in the key agreement with TTP are the same as those in the key agreement without TTP. As shown in Figure 4, we only describe the key sharing phase.

![Figure 4. Key sharing phase in the key agreement with TTP.](image)

(Step 1) If the client $ID_i$ wants to share the secret key with the cloud server $Ser_j$, the client sends a request $\{ID_i, Ser_j\}$ to TTP.

(Step 2) TTP forwards this request to the cloud server $Ser_j$.

(Step 3) The server responds a positive acknowledgement $ACK$ to TTP for agreeing the request.

(Step 4) TTP generates a symmetric bivariate polynomial $F(x, y) = a_{00} + a_{01}x + a_{10}y + a_{11}xy$, where $a_{00} = K_i$, $a_{01} (= a_{10})$, and $a_{11}$ are randomly generated. Then, TTP computes $S_c = (X_c,f_c(y))$ and $S_s = (X_s,f_s(y))$.

(Steps 5 and 6) TTP sends $\{S_c, h(K_i)\}$ and $\{S_s, h(K_i)\}$ to the cloud server and the client, respectively, through a secure channel.

4. Multi-Server Environment

A cloud service provider may build appropriate multi-server systems, and provides different services to clients. Each multi-server system contains multiple servers, which collaborate to provide various services. Suppose that $M$ servers (say Server #1, Server #2, ..., Server #M) are devoted to serve a service application. If we use SCC approaches in Figure 2 and Figure 3, respectively, to perform authentication and key recovery, the client will repeat $M$ times mutual authentication, and apply $M$ times of key recoveries to share $M$ different secret keys with $M$ servers, respectively. The proposed multi-server environment SCC (MSCC) takes the advantage of summation homomorphism property of polynomial based secret sharing to efficiently finish authentication and key recovery, respectively, by a single operation.

Consider the case of multi-server environment that includes $N$ servers, $Ser_j$, $1 \leq j \leq N$, and $n$ clients, $ID_i$, $1 \leq i \leq n$. Some notations in SCC should be modified for use in MSCC. For example, the share of client $S_c$ is modified as the share of client $ID_i$ with the corresponding cloud server $Ser_j S_c^{(i,j)} = (X_c^{(i,j)}, f_c^{(i,j)}(y))$. The selected polynomial between $ID_i$ and $Ser_j$ is $F^{(i,j)}(x, y)$, and $f_c^{(i,j)}(y) = F^{(i,j)}(X_c^{(i,j)}, y)$, where $X_c$ used in SCC is changed to $X_c^{(i,j)}$. For the share of server, the notation $S_s$ is modified as $S_s^{(i,j)} = (X_s^{(i,j)}, f_s^{(i,j)}(y))$, where $f_s^{(i,j)}(y) = F^{(i,j)}(X_s^{(i,j)}, y)$. Also, the notation $K_{i,j}$ denotes the secret key shared between $ID_i$ and $Ser_j$.

Before describing the proposed MSCC, we first prove that the symmetric property of bivariate polynomial has homomorphism property (Lemma 1), and that bivariate polynomial based secret sharing also has homomorphism property (Lemma 2). These two homomorphism properties are the whys our MSCC can reduce the operations in a multi-server environment.
Lemma 1: If \( F^{(ij)}(x, y) \), \( 1 \leq j \leq M \), is symmetric, then \( F_M(x, y) = \sum_{j=1}^{M} F^{(ij)}(x, y) \) is symmetric.

Proof: We want to prove that the polynomial \( F_M(x, y) \) is symmetric, i.e., \( F_M(x, y) = F_M(y, x) \). Because \( F^{(ij)}(x, y) \) is symmetric, it implies \( F^{(ij)}(x, y) = F^{(ij)}(y, x) \). Therefore, we have \( F_M(x, y) = \sum_{j=1}^{M} F^{(ij)}(x, y) = \sum_{j=1}^{M} F^{(ij)}(y, x) = F_M(y, x) \). The proof is completed. \( \square \)

Lemma 2: Suppose that \( M \) secret sharing schemes are constructed by polynomials \( F^{(ij)}(x, y) \), \( 1 \leq j \leq M \), respectively. The homomorphism property implies that the additive sum of shares of \( F^{(i,1)}(x, y), F^{(i,2)}(x, y), \ldots, F^{(i,M)}(x, y) \) (i.e., \( F^{(i,1)}(x_i, y_i) + F^{(i,2)}(x_i, y_i) + \cdots + F^{(i,M)}(x_i, y_i) \)) is the share of secret sharing constructed from the additive sum of polynomials \( F_M(x, y) = \sum_{j=1}^{M} F^{(ij)}(x, y) \).

Proof: Construct a secret sharing scheme by \( F_M(x, y) = \sum_{j=1}^{M} F^{(ij)}(x, y) \), and a share of this secret sharing scheme is \( F_M(x_i, y_i) \). Since \( F_M(x, y) = \sum_{j=1}^{M} F^{(ij)}(x, y) \), we have \( F_M(x_i, y_i) = \sum_{j=1}^{M} F^{(ij)}(x_i, y_i) \). The proof is completed. \( \square \)

The above homomorphism property of bivariate polynomial is similar to the homomorphism in [31]. Additionally, there were some researches [32, 33] dedicated to authentication problem for multi-server. In this paper, we adopt the summation homomorphism of polynomial to easily and quickly finish authentication for multi-server environment in the proposed MSCC.

As we know, the difference between SCC with TTP and SCC without TTP is the way of generating polynomial \( F(x, y) \). Here, we only show MSCC with TTP to describe key sharing phase, mutual authentication phase, and key recovery phase.

**Key Sharing Phase:**

All steps in MSCC are same as those in Figure 4. Additionally, every \( ID_i \) selects a common number \( X_c^{(i)} \) for all cloud servers, and then computes the multi share \( S^{(1)} = (X_c^{(i)}, f_1^{(ij)}(y)) \), where \( f_1^{(ij)}(y) = F^{(ij)}(X_c^{(i)}), y) \). Also, all servers select a common number \( X_s^{(i)} \) for the client \( ID_i \) and then compute the multi share \( S^{(2)} = (X_s^{(i)}, f_2^{(ij)}(y)) \), where \( f_2^{(ij)}(y) = F^{(ij)}(X_s^{(i)}), y) \). Finally, both sides store multi shares \( S^{(1)} \) (client) and \( S^{(2)} \) (server). Figures 5(a) and (b) are the contents stored in the client \( ID_i \) and the cloud server \( Ser_j \), respectively.

![Figure 5](image)

**Mutual Authentication Phase:**

Suppose that the client \( ID_i \) wants to log in a multi-server system including \( M \) servers (say \( Ser_1, Ser_2, \ldots, Ser_M \)). The common numbers \( X_c^{(i)} \) and \( X_s^{(i)} \) are used in MSCC to achieve the mutual authentication for these \( M \) servers simultaneously in one authentication operation. All steps in this mutual authentication are shown in Figure 6, and the details are briefly described below.

![Figure 6](image)

(Step 1) The client sends \( \{ID_i, X_c^{(i)}\} \) to the cloud server.
(Step 2) The cloud servers \((Ser_1, Ser_2, \ldots, Ser_M)\) use \(ID_i\) to find the corresponding \(f_2^{(i,1)}(y), f_2^{(i,2)}(y), \ldots, f_2^{(i,M)}(y)\), and compute \(F_2(y) = \sum_{j=1}^{M} f_2^{(i,j)}(y)\). Then, determine \(K_{int}\) from \(K_{int} = h(F_2(X_s^{(i)}))\).

(Step 3) The cloud server sends \(X_s^{(i)}\) to the client.

(Step 4) The client also obtains \(K_{int}\) from \(h(F_1(X_s^{(i)}))\), where \(F_1(y) = \sum_{j=1}^{M} f_1^{(i,j)}(y)\). Because \(F_1(X_s^{(i)}) = F_2(X_s^{(i)})\) (see Theorem 1), the client and the cloud server can share the same intermediate key \(K_{int}\).

(Step 5 – Step 9) Same as the steps in SCC.

**Theorem 1:** The values \(F_1(X_s^{(i)})\) and \(F_2(X_s^{(i)})\) used in MSCC are equal.

**Proof:** Because \(F_1(X_s^{(i)}) = \sum_{j=1}^{M} f_1^{(i,j)}(X_s^{(i)})\) and \(f_1^{(i,j)}(y) = F^{(i,j)}(X_s^{(i)}, y)\), we have \(F_1(X_s^{(i)}) = \sum_{j=1}^{M} F^{(i,j)}(X_c^{(i)}, X_s^{(i)})\). Additionally, because \(F_2(X_s^{(i)}) = \sum_{j=1}^{M} f_2^{(i,j)}(X_c^{(i)})\) and \(f_2^{(i,j)}(y) = F^{(i,j)}(X_s^{(i)}, y)\), we have \(F_2(X_c^{(i)}) = \sum_{j=1}^{M} F^{(i,j)}(X_s^{(i)}, X_c^{(i)})\). From Lemma 1, we have \(\sum_{j=1}^{M} F^{(i,j)}(X_s^{(i)}, X_c^{(i)}) = M\sum_{j=1}^{M} F^{(i,j)}(X_s^{(i)}, X_s^{(i)})\) and imply \(F_1(X_s^{(i)}) = F_2(X_c^{(i)})\). $\square$

**Key Recovery Phase:**

In our MSCC, multiple servers collaborate together to provide services. After the key recovery phase, these multiple servers can share a common secret key with the client to secure the transmission and protect data privacy.

All steps in key recovery phase are shown in Figure 7, and the details are briefly described below.

![Figure 7. Key recovery phase in MSCC.](image-url)
Proof: Obviously, we can generate two shares of $F(i,j)(x, y)$ by selecting $x = X_c(i)$ and $X_s(i)$. These two shares are $(X_c(i), F(i,j)(X_c(i), y))$ and $(X_s(i), F(i,j)(X_s(i), y))$. Lemma 2 implies that the additive sum of shares, $(X_c(i), \sum_{j=1}^{M} F(i,j)(X_c(i), y))$

and $(X_s(i), \sum_{j=1}^{M} F(i,j)(X_s(i), y))$ are two shares of additive sum of polynomials $\sum_{j=1}^{M} F(i,j)(x, y)$. Since $S_1 = (X_c(i), \sum_{j=1}^{M} F(i,j)(y)) = (X_c(i), \sum_{j=1}^{M} F(i,j)(X_c(i), y))$

and $S_2 = (X_s(i), \sum_{j=1}^{M} F(i,j)(y)) = (X_s(i), \sum_{j=1}^{M} F(i,j)(X_s(i), y))$

(see (Step 5–1) and (Step 5–2)). We can reconstruct the polynomial $F_m(x, y) = \sum_{j=1}^{M} F(i,j)(x, y)$, and thus $F_m(0, 0) = \sum_{j=1}^{M} F(i,j)(0, 0) = \sum_{j=1}^{M} K_{ij}$. □

5. Performance, Security Analysis, and Applications

5.1. Performance Evaluation

Next, we discuss the following issues to compare Yeh’s PASS, the proposed SCC, and the proposed MSCC in detail.

 Shares sent from server and client:

Although Yeh’s PASS [17] also uses secret sharing approach, its univariate polynomial does not have the symmetric property. Via the symmetric property of bivariate polynomial, our SCC and MSCC can share an intermediate key $K_{int}$ between the client and the cloud servers. The client can use the symmetric encryption (e.g., AES) to send his share to the cloud server (i.e. the client sends $E_{K_{int}}(S_1)$ to the cloud server). Also, the cloud server can send $E_{K_{int}}(S_2)$ to the client. However, in Yeh’s PASS, the client needs to use the public key of AS ($D_A$) to send its share $ENC_{D_A}(client’s \ counter \ || \ CS_i)$ to AS. Instead of using public cryptosystem, our scheme only uses symmetric encryption to transmit share between AS and the client. Finally, we save the encryption/decryption cost.

Secret key:

A weakness of PASS using public cryptosystem to send the share is that the cloud server cannot send the server’s share $SS_i$ to the client. All clients know the public key of AS is reasonable, but AS knows public keys of all clients in cloud system are hard to achieve. Therefore, in PASS, the cloud server cannot send the share to the client by using public key cryptosystem. This causes that the client cannot recover the secret key from shares. Encryption/decryption in Yeh’s PASS should be accomplished with the help of ES and QS. If we easily store the secret key in the client side, the secret key will be leaked out when the client’s device is compromised. For example, the local computer or the smart card is cracked. In our schemes, the cloud server can also send its share using the intermediate key, and thus the client can recover the secret key using his own share and the received share from the cloud server. Therefore, the client does not need to store the secret key.

Authentication cost:

In Yeh’s PASS, AS decrypts $CS_i$ and then uses the stored $SS_i$ to recover the secret key $K_i$ by Shamir’s $(2, 2)$ secret sharing, as described in (Step 5) of Secret Key Agreement. By checking the hash value $h(K_i)$, AS can verify the client. Our scheme adopts the symmetric property of $F(x, y)$ (i.e., $f_i(X_c) = f_c(X_s)$) to share the intermediate key, on which the authentication can be easily accomplished.

Mutual Authentication:

By the same intermediate key and using challenge/response handshake, our scheme can achieve mutual authentication. However, the PASS can only finish the authentication of client.

Homomorphism property in MSCC:

In the present cloud environment, some applications may need different collaborated servers. When directly applying SCC for multi-server environment, the authentication should be repeated $M$ times for a multi-server system including $M$ servers. Via the homomorphism property,
our MSCC can authenticate $M$ servers simultaneously in one operation. For key recovery, if using SCC for multi-server environment, we need $M$ secret keys for these $M$ servers. In MSCC, the homomorphism property lets the client share one common secret key with these $M$ servers.

According to the above discussions, the comparison among PASS, SCC and MSCC is summarized in Table 2.

### 5.2. Security Analysis

Our SCC has two types: one is with TTP and the other is without TTP. It is reasonable that TTP is assumed to be honest and is trusted by the client and the cloud server. For the SCC without TTP, we adopt ECDH to securely share the bivariate polynomial. Both types assure of securely sharing bivariate polynomial between the client and the cloud server. The main objective of the proposed SCC/MSCC is to prevent malicious insiders in cloud server and outsiders to login the authorized account and determine the secret key.

We first define the scope of the security issues that our SCC and MSCC discuss: (i) outsider attack, (ii) insider attack. (iii) server side attack, and (iv) client side attack. Here, we use outsider and insider to represent the attacker who is unauthorized and authorized to the cloud server. For example, a hacker in the Internet is an outsider, while a malicious cloud employee is an insider.

The difference between the protocols with TTP and without TTP is only the generation of bivariate polynomial. So, we only give security analysis for SCC and MSCC for each security issue.

**Outsider Attack:**

An attacker from outside the perimeter is not authorized to access the cloud database. He can only intercept the information from the public channel, i.e., can only collect $x$-coordinates of the shares for the client and the cloud server.

**SCC:** As shown in Figure 2, the outsider can obtain the $x$-coordinate of client’s share $X_c$ in (Step 1) and the $x$-coordinate of client’s share $X_s$ in (Step 3). However, he does not have $f_c(y)$ and $f_s(y)$, and thus he cannot recover the intermediate key $K_{int} = h(f_c(X_s)) = h(f_s(X_c))$. Because the shares and random number are encrypted by $K_{int}$, the malicious attacker cannot obtain any information.

**MSCC:** All steps in MSCC are same as those in SCC except that every client $ID_i$ selects a common $x$-coordinate $X_{c(i)}$ for $M$ cloud servers and cloud servers also select a common $x$-coordinate $X_{s(i)}$ for the client $ID_i$. By the same approach in SCC, we use the symmetric property to determine $K_{int} = h(F_2(X_{c(i)})) = h(F_1(X_{s(i)}))$ (see (Step 2) and (Step 4) in Figure 6). However, the attacker can get the $X_{c(i)}$ and $X_{s(i)}$ from public channel, he does not have $F_1(y)$ and $F_2(y)$. Thus, attackers cannot recover the intermediate key. Same as SCC, the shares and random number are encrypted by $K_{int}$, finally the malicious attacker cannot obtain any information.

<table>
<thead>
<tr>
<th>PASS</th>
<th>the proposed scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCC</td>
<td>MSCC</td>
</tr>
<tr>
<td>share sent from server</td>
<td>using public key cryptosystem</td>
</tr>
<tr>
<td>share sent from client</td>
<td>NO</td>
</tr>
<tr>
<td>secret key</td>
<td>stored in client side</td>
</tr>
<tr>
<td>authentication cost</td>
<td>using secret sharing</td>
</tr>
<tr>
<td>mutual authentication</td>
<td>NO</td>
</tr>
<tr>
<td>multi-server environment</td>
<td>applying operation for each server</td>
</tr>
</tbody>
</table>

Table 2. Comparison of Cloud Computing Schemes.
**Insider Attack:**

Suppose that malicious cloud employees can effectively acquire the access to an authorized database. So they can obtain the contents in Figure 5(b), i.e., they have server’s shares.

**SCC:** The share of the client $i$ and the server $j$ in SCC is $S_{1}^{(i,j)}$. Although the insider has $S_{1}^{(i,j)}$, he does not have the client’s share $S_{c}^{(i,j)}$. Due to the threshold of secret sharing, the malicious cloud employee cannot recover the polynomial $F^{(i,j)}(x,y)$. So, he cannot get the intermediate key (using the symmetric property of polynomial) and the secret key (using the constant term of polynomial). The insider can get the hash value $h(K_{i,j})$ in database, but it is useless to recover the secret key. On the other hand, the data of client is encrypted and protected by the secret key (see (Step 10) in key recovery phase of SCC). Finally, even though the malicious cloud employee can access the server, he gains nothing.

**MSCC:** The insider can get the share $S_{2}^{(i,j)}$ from each server, and thus he can obtain the multi share $S_{2}^{(i,j)}$ used in MSCC. However, he does not have the multi share $S_{1}^{(i,j)}$ in client side. So, the malicious cloud employee cannot recover the polynomial $F^{(i,j)}(x,y)$ in MSCC. Thus, same as SCC, the insider cannot get the intermediate key and the secret key. Moreover, the hash value of secret key $h(K_{i,j})$ in database is useless to recover the secret key $K_{i} = \sum_{j=1}^{M} K_{ij}$. The data of client can be encrypted and protected by the secret key ((Step 11) in key recovery phase of MSCC).

**Server Side Attack:**

The server side attack is defined as that the authorized cloud database is cracked by illegitimate users. Therefore, the contents in database (see Figure 5(b)) are revealed. The analysis of server side attack is same as that in insider attack. Illegitimate users gain nothing even though the cloud server is compromised.

**Client Side Attack:**

Client side attack implies that the local devices (computer, smart phone, IC card, etc.) are cracked.

**SCC:** The attacker compromises the client side and gets the client’s share $S_{c}^{(i,j)}$ and the hash value $h(K_{i,j})$ in client side (see Figure 5(a)). However, the attacker does not have the share $S_{s}^{(i,j)}$, and thus he cannot recover the polynomial $F^{(i,j)}(x,y)$ due to the threshold of secret sharing. The analysis of MSCC is similar.

**Man-in-the-Middle Attack:**

The so-called man-in-the-middle attack [34] is that attackers can intercept messages and then either relay this message or substitute another message. Because ECDH algorithm is vulnerable to the man-in-the-middle attack, thus our SCC without TTP based on ECDH may also be vulnerable to such attack. In fact, this vulnerability comes from that server and client do not use identity authentication for each other in Figure 1. It can be overcome with the use of digital signature and public-key certificate. Here, we adopt the approach of identification authentication in [35] to resist man-in-the-middle attack in our scheme.

Our identification authentication is based on that server has saved a hash value of temporary $ID_{i}(TID_{i})$ for each client $ID_{i}$. Note: this $TID_{i}$ is effective for a period, and should be renewed at the end of its lifetime. Also, the client has his own $TID_{i}$. The concept of $TID_{i}$ is similar to the temporary mobile station identification (TMSI) in mobile communication protocol.

Steps (1)–(4) in Figure 1 are modified as the following five steps in Figure 8 to resist man-in-the-middle attack.

(Step 1) Client provides a service request.

![Figure 8. Resistant to man-in-the-middle attack using identity authentication.](image-url)
(Step 2) Cloud server selects a random $R_s$ and sends it to the client.

(Step 3) Client randomly selects $n_c$ and calculates $P_c = n_c \times G$. Also, he uses his own $TID_i$ to calculate $h(h(TID_i)||R_s||P_c)$. Then, he selects a random $R_c$, and sends $(P_c, h(h(TID_i)||R_c||P_c), R_c)$ to the cloud server.

(Step 4) Cloud server selects $n_s$ and calculates $P_s = n_s \times G$. Also, it uses $h(TID_i)$ to calculate $h(h(TID_i)||R_c||P_s)$. Then, it sends $(P_s, h(h(TID_i)||R_c||P_s)$ to the client.

(Steps 5) Both sides verify whether the hash value is correct or not. If the verification is correct, the client and the cloud server calculate a point $Q$ on elliptic curve by $n_c \times P_s$ and $n_s \times P_c$, respectively.

### 5.3. Applications

In this paper, we propose two types of secure cloud computing: one is SCC and the other is MSCC. Also, the key sharing in both SCC and MSCC can be implemented with using TTP and without using TTP, respectively.

Here, we show two scenarios where our schemes can be applied: (i) single server or multi server (SCC or MSCC) (ii) using TTP or not using TTP (with TTP or w/o TTP) for key sharing.

**SCC or MSCC:**

In SCC, a cloud server can provide services to customers alone. For some applications, the services need to be accomplished through different servers. In MSCC, a service provider may organize resources and builds appropriate multi-server systems to provide various services to customers. Each multi-server system contains multiple servers, which can be devoted to serve one type of service requests and applications. However, more servers increase authentication cost when using SCC approach. Our MSCC is based on SCC, and uses the same approach through summation homomorphism. Therefore, when customers submit service requests to a service provider, the service provider determines adopting SCC (single server) or MSCC (multiple servers) for providing services.

As we know, everything on one server (single-server environment) is easy for setting up an application quickly. However, it offers little in the way of scalability and component isolation. Suppose that application and database contend for the same server resources. This case may cause poor performance. To prevent this problem, a multiple-server environment is the most common application scenario. For example, we may install Microsoft Internet Information Services (IIS) and Microsoft SQL Server on different computers. On the other hand, cloud computing is a large-scale distributed computing paradigm where computing resources are available to users. Therefore, a multi-server environment has the good scalability.

Here, we use an example of integrating multi servers to improve the performance in cloud computing [36] to demonstrate our advantage. For example, we can adopt the approach of using load balancers to implement the server setups in cloud computing. Via distributing the workload across multiple servers, we can enhance not only the performance but also the reliability. When one of the servers fails, other servers will handle the traffic until it recovers from a server failure.

**With TTP or w/o TTP:**

When using TTP to implement the key sharing phase, we need a secure channel, e.g., VPN, and this enhances the transmission cost. Also, we need a third party in SCC/MSCC. If we use ECDH in key sharing phase, we do not need TTP. However, Diffie Hellman-like protocol will be compromised by the so-called clogging attack, in which an opponent sends a public Diffie Hellman key to the AS. The AS then computes the secret key. Repeated messages of this type can clog AS with useless work. As a result, AS spends considerable computing resources for doing useless computation.

### 6. Conclusion

In this paper, we propose two types of SCC: one is with TTP and the other is without TTP. The main objective of our schemes is to protect the data privacy and security in the cloud server. We add the symmetric property in secret sharing to successfully reduce the cost to share the shares between the client and the server. Additionally, by the homomorphism property of polynomial based secret sharing, we extend
SSC to MSCC, fitting the multi-server environment. When compared with the previous PASS, our schemes have the better security and performance.

Acknowledgment

This research was supported in part by the Ministry of Science and Technology under Grant 102-2221-E-259-009-MY2 and 103-2221-E-259-015.

References


