A Server-Aided Computation Protocol Revisited for Confidentiality of Cloud Service

Yoshiaki Shiraishi
Nagoya Institute of Technology
Aichi, Japan
zenmei@nitech.ac.jp

Masami Mohri
Gifu University
Gifu, Japan
mmohri@gifu-u.ac.jp

Youji Fukuta
Aichi University of Education
Aichi, Japan
yfukuta@auecc.aichi-edu.ac.jp

Abstract

In cloud-computing services, using the SSL/TLS protocol is not enough to ensure data confidentiality. For instance, cloud service providers can see the plaintext after the decryption at the end point of a secure channel. It is wise to introduce an encryption layer between the service client and the communication channel so the data will not be seen by the cloud service provider. The encryption/decryption process should be light for cases where a cloud-service user has a low-power device such as a smart phone. We pay attention to server-aided computation as an approach of speeding up cryptographic processing. On the other hand, for future cloud services, homomorphic encryption is a useful primitive for cryptographic protocols. In this paper, we propose a server-aided computation protocol using ElGamal encryption, which is homomorphic. The proposed protocol is secure under the discrete logarithm assumption for passive and active attacks. Furthermore, we present experimental results suggesting that the processing time of the proposed protocol is shorter than the original ElGamal encryption.

1 Introduction

The cloud-computing environment, a large-scale distributed system, has been realized and is spreading rapidly. However, one of the biggest problems faced by cloud computing is security [1]. In particular, among the ten disincentives in cloud computing [2], data confidentiality is listed as a major security issue.

Data encryption with proper key management is a typical approach for a cloud user to keep data confidential. In general Web services, the SSL/TLS protocol is a measure used to ensure confidentiality from attackers on a communication channel. On the service server, however, the encryption channel is terminated, so the service provider can read the plaintext freely. In the client/server model, because there is a one-to-one mapping between a service provider and the user to whom he/she wants to send data, this situation does not arise. In cloud computing, since the front-end service server is different from the data-store server and the user cannot know which server keeps his/her data in the cloud, confidentiality must be secured in a higher layer than SSL/TLS communication.

However, we must consider what should be used for terminals with lower performance than desktop PCs, such as smart-phones. There are several encryption algorithms used in layers higher than SSL/TLS; however, it is impractical to perform these functions in a script language on a Web browser, and encryption generally requires dedicated hardware.

This paper proposes a server-aided computation protocol with ElGamal encryption, which is light enough to use on a Web browser in a layer higher than SSL/TLS. In future high-level cloud services,

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 2, number: 2, pp. 83-94

homomorphic cryptosystems such as ElGamal encryption play an important role as they can use a statistical process on ciphertexts. This paper shows that the proposed protocol is secure under the discrete logarithm assumption for passive and active attacks. Finally, we show through experimental results that the processing time of the proposed protocol is shorter than original ElGamal encryption.

2 Securing Cloud Service Confidentiality Using Conventional Techniques

In cloud computing, there are three entities: First is the “cloud service-user”; second is the “cloud infrastructure provider,” which runs the data center; and last is the “cloud service-provider,” which provides various (compound) services in the cloud. The cloud infrastructure provider and service provider together are called the “cloud provider” [3].

It is common in current Web services to secure data on a communication channel to prevent eavesdropping or falsification. SSL/TLS is widely used as the prevention method, but it is not sufficient in the case of cloud services.

In the case that a cloud service-user sends data to the cloud provider who processes it and replies to the request, as shown in Figure 1, if the data is protected by SSL/TLS, it cannot be eavesdropped upon or falsified. However, the cloud provider can view the decrypted content at the terminal point of the channel. It is better to store encrypted data in a cloud because a cloud-service user cannot know whether the infrastructure is a threat or how to counter it. Considering the above, to secure confidentiality in cloud services, it is important that the encryption be applied not to the communication channel but by the cloud-service user.

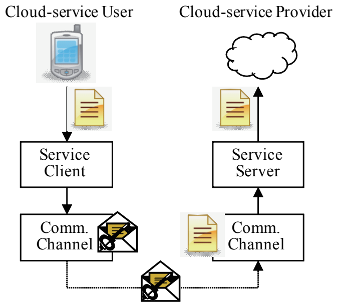

Figure 1: Data encryption by SSL/TLS protocol.

3 Revisiting Server-Aided Computation for Cloud-Service Confidentiality

3.1 Encryption Process on a Web Browser

As shown in Figure 2, an encryption layer is inserted between the communication channel and the service client so that users send and receive data, which cannot be seen by the cloud service provider (i.e., maintaining data confidentiality from the cloud-service provider). Many cloud services are provided through a Web interface; users may access it from PCs with different specifications, that is, low memory, poor CPU, and so on. Therefore, encryption and decryption are required after the data are inputted on the Web browser and after the data are received through the channel, respectively. In [4], an evaluation of several encryption/decryption libraries for Web browsers has been presented, namely, JavaScrypt [5], a hash algorithm of JavaScript [6], the JavaScript Crypt Library [7], RSA encryption using the jsbn library by Wu et.al [8], and jCryption [9]. In this evaluation, the encryption processing speeds are shown for three different browsers (Google Chrome, Internet Explorer, and Firefox) on a high-spec PC (2.66GHz Intel Xeon 5150). The results of mobile computers or smart-phones (iPod touch, etc.) are only shown as scalar multiplication on an elliptic curve.

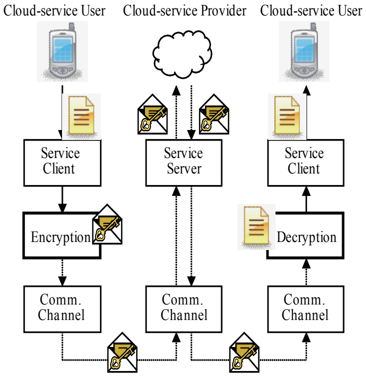

Figure 2: Data encryption on service clients.

In Table 1, for example, we compare the processing speed of ElGamal encryption / decryption [10] on a desktop PC and a smart-phone under the same conditions, that is, the same cloud service servers are used. The desktop PC has a 2.33GHz Core2 DuoCPU, 2GB of RAM, and is running the 32-bit Windows Vista Business operating system; the Web browser is Google Chrome 3.0.195.33. The smart-phone has a 624MHz Marvell Monahans PXA310 CPU with 128MB of RAM, running AndroidOS1.5; the Web browser is Chrome Lite. The server has a 3.2GHz Pentium4 with 2GB of RAM, running CentOS5.4. The key length of the ElGamal encryption is 1,024 bits, and it is implemented using Baird's BigInt library [11]. The results listed in Table 1, show that the smartphone encryption/decryption-speed is very low.

Table 1: Processing time of ElGamal encryption/decryption (key length 1,024bits) by desktop PC and smart phone (ms).

	Desktop PC	Smart-phone
Encryption	788	265,020 (4.4 min)
Decryption	424	135,820 (2.3 min)

3.2 Approach for Maintaining Confidentiality on Cloud Services

It is desirable that same encryption method be used on the mobile phone as on the desktop PC to improve the convinience for cloud users and expand the viability of cloud-service providers. As shown in Table1, in the case of a low-spec computer such as a smartphone, the time needed for encryption/decryption must be reduced. Methods to improve the encryption/decryption speed are as follows: 1) using a high-spec CPU; 2) improving the speed by software programming techniques, and 3) using a server-aided computation method–part fo the encryption/decryption processing is done by the server. However, regarding 1), it is not feasible to achieve a CPU speedup of more than 100 times; and 2), future architectures will require custom software. In this paper, we focus on a server-aided computation technique to speed up encryption processing on a universal platform.

3.3 Conventional Methods of Server-Aided Computation

When low-spec computer A encrypts its secret to send it to another computer, it needs a lot of processing time. It is possible to shorten A's processing time by asking high-spec computer B (server) to do a part of its calculation without showing A's secret. This method is called server-aided computing. A server-aided computing protocol using RSA encryption has been shown in [12].

Attacks against server-aided computing are classified as passive attacks [13], [14] and active attacks [13], [14], [15], [16], [17]. In a passive attack, the attacker tries to compute the client's secret by obtaining information sent from the client to a server, and the public key, via the regular protocols of server-aided computation. In an active attack, the server is assumed to be an attacker; that is, it's a malicious server that does not follow the protocol. It sends the client invalid data to presuade it to provide convenient data for the malicious server.

Several server-aided computation protocols which can prevent passive attacks [18], [19] and many protocols which have a check function to prevent active attacks [15], [19], [20], [21], [22], [23] have been shown.

3.4 Protocols using homomorphism

Homomorphic encryption is used as the basic encryption technique in future services such as e-voting / questionnaires [24], [25], [26], electronic cash [27], and electronic auctions [28], [29]. We anticipate that many cloud services will appear, which can use the user's encrypted data stored on a cloud server without decryption. In such cases, it is better to have a Web browser compute the encryption process using homomorphism. However, as shown in Sec.3.3, although there are several studies of server-aided computation using RSA encryption, we have never come across a study of it using homomorphism. We propose a protocol for server-aided computation using ElGamal encryption, which is a class of homomorphic encryption.

4 Proposed Protocol: Server-Aided ElGamal Encryption

Modular exponentiation is the arithmetic operation, which has the highest cost in the ElGamal encryption/decryption algorithm. ElGamal encryption, classified as probabilistic encryption, calculates the modular exponentiation for random numbers per session. If somebody (an attacker) can determine the random number for a modular exponentiation, he/she can decrypt the ciphertext without the private key. Therefore, we propose a new server-aided computation protocol. Using this protocol, users can encrypt and decrypt in a short processing time without losing the confidentiality of data (random number) to the cloud service server.

U_1 and U_2 denote cloud-service users. Here, U_1 (U_2) is a sender (receiver). S is the server for server-aided computation and T is a storage server. The proposed protocol is as follows:

[Preprocess]
U_2 generates a secret key/public key pair $(sk, pk) = (x, (p, g, y = g^x))$, where p is prime and g denotes a generator element of $Z_p^* = \{1, 2, \cdots, p-1\}$.

[Encryption process] (Figure.3)

Step1) $m \in Z_p$ denotes U_1's message. U_1 generates two small random numbers $r_{(U_1,1)}$, $r_{(U_1,2)}$. Here, $r_{(U_1,1)}$ denotes a power value of modular exponentiation, which is used by U_1 in the encryption process. $r_{(U_1,2)}$ denotes a power value of modular exponentiation used by U_1 for checking the result. Calculate $r_{(U_1,3)}$ such that $p-1 = r_{(U_1,2)} + r_{(U_1,3)}$ from $r_{(U_1,2)}$ and p. Then using $r_{(U_1,1)}$ and $r_{(U_1,3)}$, calculate $r = r_{(U_1,1)}r_{(U_1,3)} \bmod p-1$, where, $r_{(U_1,3)}$ is a power value that is sent to S for server-aided computation. U_1 sends $r_{(U_1,3)}$ to S.

Step2) After receiving $r_{(U_1,3)}$ from U_1, S calculates the temporary cipher text $(Z_{(U_1,1)}, Z_{(U_1,2)})$ using $r_{(U_1,3)}$, p, g, and y, $(Z_{(U_1,1)}, Z_{(U_1,2)}) = (g^{r_{(U_1,3)}} \bmod p, y^{r_{(U_1,3)}} \bmod p)$. S sends $(Z_{(U_1,1)}, Z_{(U_1,2)})$ to U_1.

Step3) After receiving temporary cipher text $(Z_{(U_1,1)}, Z_{(U_1,2)})$ from S, U_1 checks following equations for $Z_{(U_1,1)}$, $Z_{(U_1,2)}$, $r_{(U_1,2)}$, p, g, and y.

$$\begin{aligned} Z_{(U_1,1)}g^{r_{(U_1,2)}} &= g^{r_{(U_1,3)}}g^{r_{(U_1,2)}} = g^{p-1} \equiv 1(\bmod p) \\ Z_{(U_1,2)}y^{r_{(U_1,2)}} &= y^{r_{(U_1,3)}}y^{r_{(U_1,2)}} = y^{p-1} \equiv 1(\bmod p) \end{aligned}$$

From the above results, if the pair of temporary cipher values $(Z_{(U_1,1)}, Z_{(U_1,2)})$ are invalid, then U_1 stops the encryption process.

Step4) For $Z_{(U_1,1)}$, $Z_{(U_1,2)}$, $r_{(U_1,1)}$, m, and p, U_1 calculates the cipher text (c_1, c_2) using the following equations.

$$\begin{aligned} c_1 &= Z_{(U_1,1)}^{r_{(U_1,1)}} \bmod p \\ &= (g^{r_{(U_1,3)}})^{r_{(U_1,1)}} \bmod p \\ &= g^r \bmod p \\ c_2 &= mZ_{(U_1,2)}^{r_{(U_1,1)}} \bmod p \\ &= m(y^{r_{(U_1,3)}})^{r}_{(U_1,3)} \bmod p \\ &= my^r \bmod p \end{aligned}$$

U_1 sends the cipher text (c_1, c_2) to T.

Step5) T receives cipher text (c_1, c_2) from U_1, and stores it. When U_2 requests U_1's cipher text from T. T sends (c_1, c_2) to U_2.

We can easily execute the calculation checking process in Step3) by using Fermat's theorem.

[Fermat's Theorem] *For any $a \in Z_p^*$, the following equation is satisfied:*

$$a^{p-1} \equiv 1(\bmod p),$$

where p is a prime number.

[Decryption process] (Figure. 4)

Step1) U_2 receives cipher text (c_1, c_2). Then, having calculated $x' = p-1-x$ for $sk(=x)$ and p, U_2 generates two random numbers, $r_{(U_2,1)}$ and $r_{(U_2,2)}$. $r_{(U_2,1)}$ and $r_{(U_2,2)}$ are exponents and used by U_2, in the decryption and calculation checking process, respectively. For $r_{(U_2,1)}$, x' and p, U_2 calculates $r_{(U_2,3)}$ such that $x' = r_{(U_2,1)}r_{(U_2,3)} \bmod p-1$. For $r_{(U_2,2)}$, $r_{(U_2,3)}$ and p, U_2 calculates $r_{(U_2,4)}$ such that $p-1 = r_{(U_2,2)} + r_{(U_2,3)} + r_{(U_2,4)}$, where $r_(U_2,3)$ and $r_{(U_2,4)}$ are exponents and aided with calculation by S. U_2 sends $r_{(U_2,3)}$, $r_{(U_2,4)}$ and a part of the ciphertext to S.

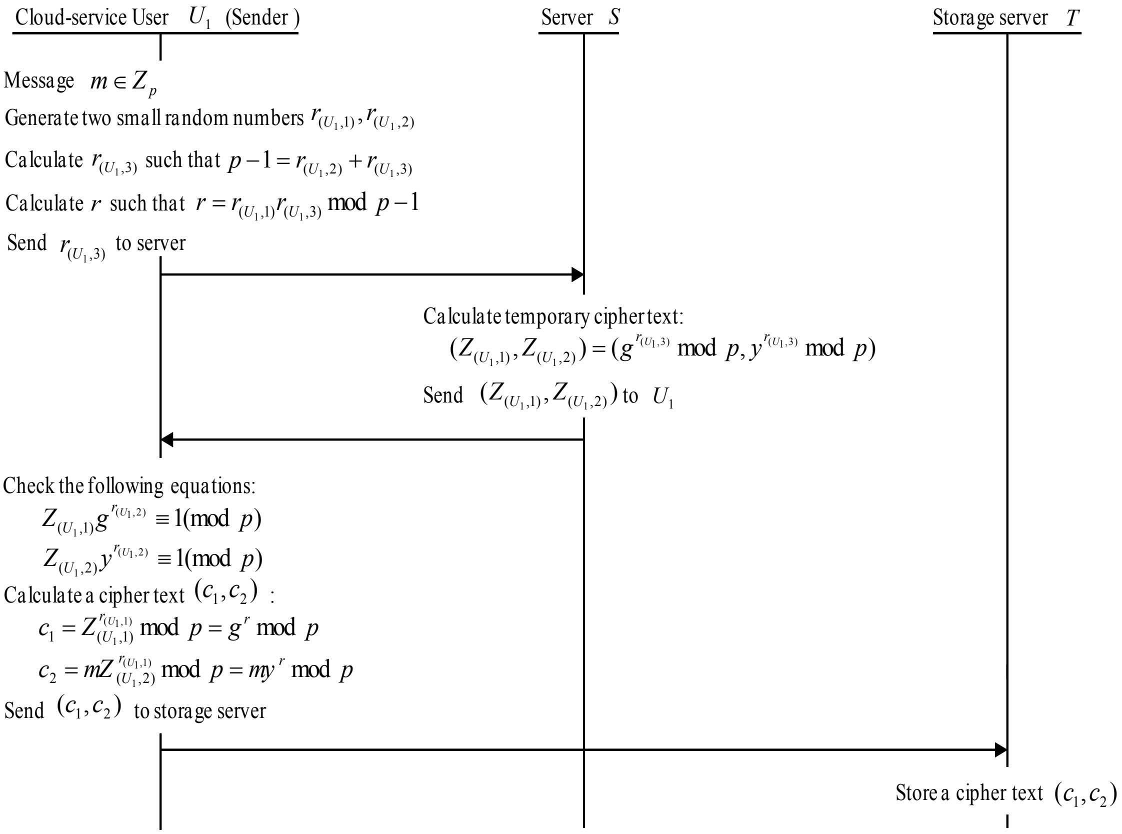
Cloud-service User U_1 (Sender)
Server S
Storage server T
Message $m \in Z_p$
Generate two small random numbers $r_{(U_1,1)}, r_{(U_1,2)}$
Calculate $r_{(U_1,3)}$ such that $p-1 = r_{(U_1,2)} + r_{(U_1,3)}$
Calculate r such that $r = r_{(U_1,1)} r_{(U_1,3)} \bmod p-1$
Send $r_{(U_1,3)}$ to server
Calculate temporary ciphertext:
$(Z_{(U_1,1)}, Z_{(U_1,2)}) = (g^{r_{(U_1,3)}} \bmod p, y^{r_{(U_1,3)}} \bmod p)$
Send $(Z_{(U_1,1)}, Z_{(U_1,2)})$ to U_1
Check the following equations:
$Z_{(U_1,1)} g^{r_{(U_1,2)}} \equiv 1 (\bmod p)$
$Z_{(U_1,2)} y^{r_{(U_1,2)}} \equiv 1 (\bmod p)$
Calculate a cipher text (c_1, c_2) :
$c_1 = Z_{(U_1,1)}^{r_{(U_1,1)}} \bmod p = g^r \bmod p$
$c_2 = mZ_{(U_1,2)}^{r_{(U_1,1)}} \bmod p = my^r \bmod p$
Send (c_1, c_2) to storage server
Store a cipher text (c_1, c_2)

Figure 3: Encryption process of the proposed server-aided computation protocol with server *S* and sender U_1.

Step2) After $r_{(U_2,3)}$, $r_{(U_2,4)}$, c_1 and *p* are received, *S* calculates the temporary ciphertext as follows:

$$(Z_{(U_2,1)}, Z_{(U_2,2)}) = (c_1^{r_{(U_2,3)}} \bmod p, c_1^{r_{(U_2,4)}} \bmod p).$$

Step3) After the temporary cipher $(Z_{(U_2,1)}, Z_{(U_2,2)})$ is received from *S*, U_2 tries to check the following equation.

$$Z_{(U_2,1)} Z_{(U_2,2)} c_1^{r_{(U_2,2)}} = c_1^{r_{(U_2,3)}} c_1^{r_{(U_2,2)}} c_1^{r_{(U_2,2)}} = c^{p-1} \equiv 1 (\bmod p)$$

From the above results, if the temporary cipher values $(Z_{(U_2,1)}, Z_{(U_2,2)})$ are invalid, then U_2 stops the decryption process.

Step4) For $Z_{(U_2,1)}$, ${}_{(U_2,1)}$, c_2 and *p*, U_2 obtains the message *m* using the following equations:

$$\begin{aligned} c_2 Z_{(U_2,1)}^{r_{(U_2,1)}} \bmod p &= c_2 (c_1^{r_{(U_2,3)}})^{r_{(U_2,1)}} \bmod p \\ &= c_2 c_1^{x'} = c_2 c_1^{p-1-x} = my^r g^{r(p-1-x)} = mg^{rx} g^{r(p-1)-rx} \\ &= mg^{r(p-1)} \bmod p \\ &= m \bmod p. \end{aligned}$$

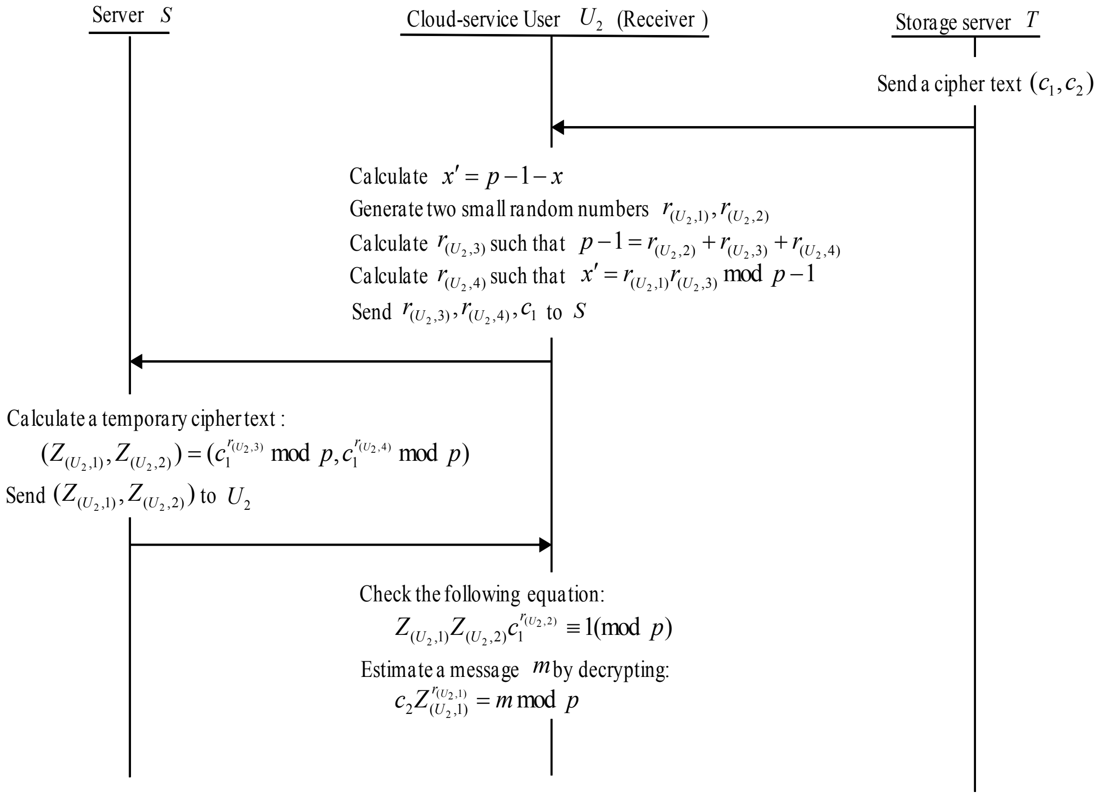
Server S
Cloud-service User U_2 (Receiver)
Storage server T
Send a cipher text (c_1, c_2)
Calculate $x' = p - 1 - x$
Generate two small random numbers $r_{(U_2,1)}, r_{(U_2,2)}$
Calculate $r_{(U_2,3)}$ such that $p - 1 = r_{(U_2,2)} + r_{(U_2,3)} + r_{(U_2,4)}$
Calculate $r_{(U_2,4)}$ such that $x' = r_{(U_2,1)} r_{(U_2,3)} \bmod p - 1$
Send $r_{(U_2,3)}, r_{(U_2,4)}, c_1$ to S
Calculate a temporary cipher text :
$(Z_{(U_2,1)}, Z_{(U_2,2)}) = (c_1^{r_{(U_2,3)}} \bmod p, c_1^{r_{(U_2,4)}} \bmod p)$
Send $(Z_{(U_2,1)}, Z_{(U_2,2)})$ to U_2
Check the following equation:
$Z_{(U_2,1)} Z_{(U_2,2)} c_1^{r_{(U_2,2)}} \equiv 1 (\bmod p)$
Estimate a message m by decrypting:
$c_2 Z_{(U_2,1)}^{r_{(U_2,1)}} = m \bmod p$

Figure 4: Decryption process of the proposed server-aided computation protocol with server S and receiver U_2.

5 Security

5.1 Definitions of Adversary and Security

The goal of adversary A is to know the secrets of sender U_1 or receiver U_2 or to falsify message m. We evaluate the proposed protocol against passive and active attacks as same as related works shown in Sec.3.3.

A passive attack is where an adversary obtains a secret from public information and transmitted data. The secrets in the proposed protocol are r and $sk(=x)$ in encryption and decryption, respectively.

An active attack is where S, who is adversary A, does not work according to the protocol, and sends falsified values to U_1 and U_2; then S obtains the secret or falsifies message m.

The protocol is secure if it is difficult for passive or active adversary A to successfully complete the above attack on U_1 or U_2.

5.2 Assumption

The security of the proposed protocol is based on the following discrete logarithm assumption.

[Discrete logarithm assumption] The discrete logarithm problem is to obtain x from the following equation:

$$a = g^x (\bmod p),$$

when a prime number p, a generator g of Z_p^*, and $a \in Z_p^*$ are given. The problem is difficult to resolve, if p is large.

5.3 Results for a Passive Attack

Adversary A cannot obtain r generated by U_1 in the following two cases: 1) for a one-round attack, p is large and $r_{(U_1,1)}$ is appropriately generated; and, 2) for a multi-round attack, p is large and $r_{(U_1,1)}$, $r_{(U_1,2)}$ are appropriately re-generated. So, A cannot get message m in the encryption process.

Adversary A cannot obtain $sk(=x)$ of U_2 in the following two cases: 1) for a one-round attack, p is large and $r_{(U_2,1)}$ is appropriately generated; and, 2) a for multi-round attack, p is large, and $r_{(U_2,1)}$, $r_{(U_2,2)}$ are appropriately re-generated. Thus, the decryption process is also secure against passive attack.

5.4 Results for an Active Attack

The goal of adversary A in the encryption process is to obtain r by using (c_1', c_2') generated from $(Z'_{(U_1,1)}$, $Z'_{(U_1,2)})$ by U_1, where $(Z'_{(U_1,1)}, Z'_{(U_1,2)})$ is sent by A. However, U_1 checks whether $(Z'_{(U_1,1)}, Z'_{(U_1,2)})$ is correct or not in Step3. Therefore, A cannot obtain the necessary data to compute r. In a multi-round attack, what A can or cannot do is the same. Therefore, the encryption process is secure against active attacks.

The goal of adversary A in the decryption process is to falsify m, that is, U_2 accepts falsified message m' computed from $(Z'_{(U_2,1)}, Z'_{(U_2,2)})$, where $(Z'_{(U_2,1)}, Z'_{(U_2,2)})$ is sent by A. However, U_1 checks whether $(Z'_{(U_1,1)}, Z'_{(U_1,2)})$ is correct in Step3. Therefore, the decryption process is also secure against passive attacks.

6 Efficiency of the Proposed Protocol

As shown in Table 1, the processing time of ElGamal encryption/decryption on a smart-phone is over several hundred seconds.

6.1 Efficiency

Table 2 shows the efficiency of ElGamal encryption/decryption during conventional browser use. The number of modular exponentiations of encryption and decryption are two and one, respectively. Since the processes are contained in the user-side, there is no process on a server.

In contrast, the efficiency of the proposed protocol is shown in Table 3. The numbers of modular exponentiation in encryption and decryption are four and two for the user, respectively. Two and one modular exponentiations are added to the server process in encryption and decryption, respectively. The communication costs are $3|p|$ and $2|p|$ bits in each process.

The proposed protocol increases the apparent cost in comparison with the original encryption and decryption. However, from the following three points, the proposed protocol completes in a shorter time than the original processing: first is that the performance of a server and JavaServlet are higher than a client and JavaScript; second is that the communication time is negligible if a user connects to the server through a high speed channel; last is that the processing time is shorter than in the original one because the exponent size of the proposed protocol is smaller.

6.2 Implementation

The proposed protocol and simple ElGamal encryption/decryption is implemented and measured from the view of processing and communication time. The measured environment is the same as Sec3.3.

Table 2: Efficiency of ElGamal encryption on user's browser.

	Modular exponentiation		Communication
	User	Server	cost
Encryption	2	—	—
Decryption	1	—	—

Table 3: Efficiency of the proposed protocol.

	Modular exponentiation		Communication
	User	Server	cost
Encryption	4 (verify: 2)	2	$3\|p\|$ bit
Decryption	2 (verify: 1)	1	$2\|p\|$ bit

The result of a simple ElGamal encryption / decryption on the user's browser with the key length $|p| = 1,024$ bits is shown in Table 4. Using the same key length, the results of the proposed protocol with modular exponent sizes $|r(U,1)|$ and $|r(U,2)|$ = 128, 256, and 512 are shown in Table5.

From Tables 4 and 5, the proposed protocol is about 4.1, 2.4, and 1.4 times faster at encryption and 3.2, 2.2, and 1.2 times faster at decryption than the original protocol, respectively.

Table 4: Processing time of ElGamal encryption on user's browser. (key length $|p| = 1,024$ bits)

	Processing time(ms)			
	User	Server	Comm.	Total
Encryption	265,020	—	—	265,020
Decryption	135,820	—	—	135,820

7 Conclusion

This paper proposes a server-aided ElGamal encryption/decryption protocol for providing encryption in a higher layer than SSL/TLS. The ElGamal cryptosystem is a basic and important primitive for future useful cloud services because it is homomorphic. The proposed protocol is secure against passive and active attacks under the discrete logarithm assumption. This paper shows by implementation the result that the protocol can take less time than the original one.

The processing time of the protocol depends on the exponent size of the modulas. Because the size relates to security, there is a trade-off between the processing time and security. This paper refers to a basic security analysis, but a more strict security evaluation and its relation between processing time and security are given as future work.

Table 5: Processing time of the proposed protocol. (key length $|p| = 1,024$ bits)

	Exponent size	Processing time(ms)			
	$\lvert r_{(U,1)}\rvert, \lvert r_{(U,2)}\rvert$	User	Server	Comm.	Total
Encryption	128 bit	57,827 (verify: 28,238)	65	7,185	65,077
	256 bit	105,338 (verify: 52,896)	65	2,594	108,245
	512 bit	192,036 (verify: 92,618)	65	3,011	195,112
Decryption	128 bit	35,291 (verify: 17,463)	66	6,500	41,857
	256 bit	59,179 (verify: 28,689)	66	2,594	61,839
	512 bit	109,041 (verify: 58,018)	66	2,518	111,625

References

[1] C. Balding, “Biggest Cloud Challenge: Security,” http://cloudsecurity.org/blog/2008/10/14/biggest-cloud-challenge-security.html, October 2008.

[2] M. Armbrust, et al., “Above the Clouds: A Berkeley View of Cloud Computing,” Electrical Engineering and Computer Sciences, University of California at Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009, http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf.

[3] N. Uramoto, “Cloud Computing: Security and Compliance Issues in Cloud Computing,” *IPSJ Magazine (in Japanese)*, vol. 50, no. 11, pp. 1099–1105, November 2009.

[4] T. Oda and G. Morohashi, “JavasCrypto,” *IEICE Technical Report (in Japanese)*, vol. 109, no. 272, pp. 75–80, November 2009.

[5] J. Walker, “JavaScrypt: Browser-Based Cryptography Tools,” http://www.fourmilab.ch/javascrypt/ (2011/2/28 accessed), December 2005.

[6] “Hash algorithm of JavaScript,” http://user1.matsumoto.ne.jp/~goma/js/hash.html (2011/2/28 accessed).

[7] “JavaScript Crypt Library,” http://www.clipperz.com/open_source/javascript_crypto_library (2011/2/28 accessed).

[8] T. Wu, “BigIntegers and RSA in JavaScript,” http://www-cs-students.stanford.edu/~tjw/jsbn/ (2011/2/28 accessed).

[9] “jCryption,” http://www.jcryption.org/ (2011/2/28 accessed).

[10] T. ElGamal, “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms,” *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, July 1985.

[11] L. Baird, “BigIntegers in JavaScript,” http://www.leemon.com/crypto/BigInt.html (2011/2/28 accessed).

[12] T. Matsumoto, K. Kato, and H. Imai, “Speeding up secret computations with insecure auxiliary devices,” in *Proc. of the 8th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO’88), Santa Barbara, California, USA, LNCS*, vol. 403. Springer-Verlag, August 1988, pp. 497–506.

[13] B. Pfitzmann and M. Waidner, “Attacks on protocols for server-aided RSA computation,” in *Proc. of the 11th annual international conference on Theory and application of cryptographic techniques (EUROCRYPT’92), Balatonfüred, Hungary, LNCS*, vol. 658. Springer-Verlag, May 1992, pp. 153–162.

[14] J. Burns and C. Mitchell, “Parameter selection for server-aided RSA computation schemes,” *IEEE Transactions on Computers*, vol. 43, no. 2, pp. 163–174, February 1994.

[15] A. Shimbo and S. Kawamura, “Factorization attack on certain server-aided computation protocols for the RSA secret transformation,” *Electronics Letters*, vol. 26, no. 17, pp. 1387–1388, August 1990.

[16] R. Anderson, “Attack on server assisted authentication protocols,” *Electronics Letters*, vol. 28, no. 15, pp. 1473–1473, July 1992.

[17] S. Kawamura, “Information leakage measurement in a distributed computation protocol,” *IEICE Transatcions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E78-A, no. 1, pp. 59–66, January 1995.

[18] T. Matsumoto, H. Imai, C. Laih, and S. Yen, "On verifiable implicit asking protocols for RSA computation," in *Proc. of Workshop on the Theory and Application of Cryptographic Techniques (AUSCRYPT'92), Gold Coast, Queensland, Australia, LNCS*, vol. 718. Springer-Verlag, December 1993, pp. 296–307.

[19] C. Lim and P. Lee, "Security and performance of server-aided RSA Computation protocols," in *Proc. of the 15th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'95), Santa Barbara, California, USA, LNCS*, vol. 963. Springer-Verlag, August 1995, pp. 70–83.

[20] S. Kawamura and A. Shimbo, "Performance analysis of server-aided secret computation protocol for the RSA cryptosystem," *IEICE Transactions*, vol. E73, no. 7, pp. 1073–1080, July 1990.

[21] T. Matsumoto and H. Imai, "How to ask and verify oracles for speeding up secret computations (Part 2)," *IEICE Technical Report*, vol. IT89-24, 1989.

[22] S. Kawamura and A. Shimbo, "A note on checking the faithfulness of the server in client-server system (II)," *IEICE Technical Report*, vol. ISEC89-17, 1989.

[23] J. Quisquater and M. Soete, "Speeding up smart card RSA computation protocols," in *Proc. of the Second International Smart Card 2000 Conference, Amsterdam, The Netherlands*. North-Holland, October 1989, pp. 199–206.

[24] J. Benaloh and M. Yung, "Distributing the Power of a Government to Enhance the Privacy of Voter," in *Proc. of the 5th annual ACM symposium on Principles of distributed computing (PODC'86), Calgary, Alberta, Canada*. ACM, August 1986, pp. 52–62.

[25] K. Sako and J. Kilian, "Secure Voting Using Partially Compatible Homomorphisms," in *Proc. of the 14th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'94), Santa Barbara, California, USA, LNCS*, vol. 839. Springer-Verlag, August 1994, pp. 411–424.

[26] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung, "Multi-Authority Secret-Ballot Elections with Linear Work," in *Proc. of the 1996 International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'96), Saragossa, Spain, LNCS*, vol. 1070. Springer-Verlag, May 1996, pp. 72–82.

[27] D. Chaum, "Online Cash Checks," in *Proc. of the 1989 Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT'89), Houthalen, Belgium, LNCS*, vol. 434. Springer-Verlag, April 1989, pp. 288–293.

[28] O. Baudron and J. Stern, "Non-interactive Private Auctions," in *Proc. of the 5th International Conference on Financial Cryptography (FC'01), Southampton, Bermuda, LNCS*, vol. 2339. Springer-Verlag, March 2002, pp. 364–377.

[29] M. Abe and K. Suzuki, "M+1-st Price Auction using homomorphic encryption," in *Proc. of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems (PKC'02), Paris, France, LNCS*, vol. 2274. Springer-Verlag, February 2002, pp. 115–124.

Yosahiki Shiraishi received B.E. and M.E. degrees from Ehime University, Japan, and Ph.D degree from the University of Tokushima, Japan, in 1995, 1997, and 2000, respectively. From 2002 to 2006 he was a lecturer at the Department of Informatics, Kinki University, Japan. Since 2006, he has been an associate professor at the Department of Computer Science and Engineering, Nagoya Institute of Technology, Japan. His current research interests include information security, cryptography, computer network, and knowledge sharing and creation support. He received the SCIS 20th Aniversary Award and the SCIS Paper Award from ISEC group of IEICE in 2003 and 2006, respectively. He is a member of IEEE, ACM, IPSJ, and a senior member of IEICE.

Masami Mohri received B.E. and M.E. degrees from Ehime University, Japan, in 1993 and 1995 respectively. She received Ph.D degree in Engineering from the University of Tokushima, Japan in 2002. From 1995 to 1998 she was an assistant professor at the Department of Management and Information Science, Kagawa junior college, Japan. From 1998 to 2002 she was a research associate of the Department of Information Science and Intelligent Systems, the University of Tokushima, Japan. From 2003 to 2008 she was a lecturer of the same department. Since 2008, she has been an associate professor at the Information and Multimedia Center, Gifu University, Japan. Her research interests are in coding theory, information security and cryptography. She is a member of IEEE and a senior member of IEICE.

Youji Fukuta received his M.E. degree from the University of Tokushima in 2002. He joined Aichi University of Education in 2005 and has been engaged in research on information security. In 2007, he received his Ph.D. degree in Engineering from the University of Tokushima. He is currently a lecturer at the Information Education Course of Aichi University of Education. He is a member of IEICE.