One-time Programs with Cloud Storage and Its Application to Electronic Money

Takuya Kitamura
University of Tsukuba
kitamura@cipher.risk.tsukuba.ac.jp

Kazumasa Shinagawa
University of Tsukuba
shinagawa@cipher.risk.tsukuba.ac.jp

Takashi Nishide
University of Tsukuba
nishide@risk.tsukuba.ac.jp

Eiji Okamoto
University of Tsukuba
okamoto@risk.tsukuba.ac.jp

ABSTRACT

A One-time Program (OTP) is a program, proposed by Goldwasser et al., in which the number of executions is restricted to once. The OTP uses a Garbled Circuit (GC), a circuit which does not leak the information except the execution result, as a building block. However, we need to use a special hardware called One-time Memory (OTM) to permit access to only one of two labels necessary for the execution of the GC. When the OTM does not exist or it is expensive, the realistic realization of the OTP is difficult. Furthermore, the OTM needs to be transported to a program executor and needs the production cost. Instead of using OTMs, we propose a way to distribute inputs to multiple cloud storages by using Shamir's secret sharing. In addition, we apply the proposed method to electronic money. Existing electronic money schemes do not prevent double-spending itself, but detect it and identify the double-spender. Therefore, one can double-spend the same electronic money illegally in a short time. Then we can detect the double-spending and identify the double-spender after delivering a product, but the double-spender can abscond with it. To tackle such a problem, we construct an electronic money scheme which prevents double-spending itself based on the OTP that generates a digital signature during a transaction. Combining the proposed electronic money scheme with an existing electronic money scheme, we can also construct another electronic money scheme which can detect double-spending even if the security of the OTP is broken.

CCS Concepts

•**Information systems** → **Digital cash;**

Keywords

One-time Programs, cloud storage, secret sharing, electronic money, double-spending

APKC'17 April 02 2017, Abu Dhabi, United Arab Emirates

DOI: http://dx.doi.org/10.1145/3055504.3055507

1. INTRODUCTION

In 1986, Yao proposed a circuit called Garbled Circuit (GC) to enable secure two-party computation. A GC does not leak the information except the execution result. A GC has many applications as a powerful framework in cryptography.

As an application of GCs, Goldwasser et al. [4] proposed One-time Programs (OTP) for the first time. We can execute an OTP with one input at any time, and the OTP does not leak the information except the execution result. However, we need to use a special hardware, which is resistant to side-channel attacks, called One-time Memory (OTM) to realize an OTP. When an OTM does not exist or it is expensive, the realistic realization of an OTP is difficult. Furthermore, OTMs need to be transported to a program executor and need the production cost. If we can enable OTPs without special hardware, we can enable a wider range of applications.

Our Contributions. Instead of using OTMs, we propose a way to enable an OTP by using cloud storages without special hardware.

Existing electronic money schemes do not prevent double-spending itself, but detect it and identify the double-spender. To tackle such a problem, we construct an electronic money scheme which prevents double-spending itself by using an OTP with cloud storages.

2. PRELIMINARIES

2.1 Garbled Circuit (GC)

A GC [7] [8] is a boolean circuit which is typically generated from a function and can evaluate the function without revealing the inputs. A GC uses random numbers called label $(\ell_i^0,\ \ell_i^1)_{1\le i\le m}$ corresponding to $\{0,\ 1\}^m$ as inputs, and outputs $\{0,\ 1\}$ in the clear eventually.

A generator of a GC prepares an arbitrary boolean circuit C. He transforms a truth table of each logic gate which constitutes C into a table like Table 1 (where $E_k(m)$ means an encrypted m by a symmetric encryption under key k), from a gate near the input. For example, when an input wire x of gate g_1 is an output wire z of AND gate g_0, $(\ell_x^0,\ \ell_x^1)$ in g_1 is $(\ell_z^{g_0(0,0)=g_0(0,1)=g_0(1,0)=0}, \ell_z^{g_0(1,1)=1})$ in g_0.

An executor of the GC decrypts four garbled values by using the inputs $(\ell_x,\ \ell_y)$. He adopts the value that can be

Table 1: Truth Table for logic gate g

input x	input y	(output z)	garbled value
ℓ_x^0	ℓ_y^0	$(\ell_z^{g(0,0)})$	$E_{\ell_x^0}(E_{\ell_y^0}(\ell_z^{g(0,0)}))$
ℓ_x^0	ℓ_y^1	$(\ell_z^{g(0,1)})$	$E_{\ell_x^0}(E_{\ell_y^1}(\ell_z^{g(0,1)}))$
ℓ_x^1	ℓ_y^0	$(\ell_z^{g(1,0)})$	$E_{\ell_x^1}(E_{\ell_y^0}(\ell_z^{g(1,0)}))$
ℓ_x^1	ℓ_y^1	$(\ell_z^{g(1,1)})$	$E_{\ell_x^1}(E_{\ell_y^1}(\ell_z^{g(1,1)}))$

decrypted correctly as the output and uses it as the input to the next gate. He can obtain the output in the clear eventually.

2.2 One-time Programs (OTP)

As an application of GCs, Goldwasser et al. [4] proposed a program in which the number of executions is restricted to once, and which is called One-time Programs (OTP).

A special secure hardware (One-time Memory; OTM) is necessary to realize an OTP. In the OTM, two values are stored and only one value can be read out. In addition, the OTM has a single tamper-proof bit to prevent data from being read out by the irregular means. We assume that the memory area which stores values has the tolerance to the side-channel attacks unless the OTM accesses it. If the OTM accesses the memory area to erase the value after the other value was read out, it will allow the side-channel attack. Therefore, to prevent an adversary from accessing the memory area, the OTM has a flag set as the safe bit instead of erasing the value. In this way, the OTM offers a required behavior safely.

The combination of a GC and OTMs becomes an OTP. The executor of an OTP picks up one of input labels $(\ell^0,\ \ell^1)$ to the GC from an OTM, so he can only obtain labels corresponding to one input. [1]

2.3 (k,n)-Secret Sharing

In 1979, Shamir [6] introduced a method, (k,n)-secret sharing, to divide a secret into n shares and reconstruct it by collecting k or more shares. Then, no one can learn any information of secret with less than k shares.

2.4 k-out-of-n OT

In Oblivious Transfer (OT) [3] [5], when the sender sends some data to the receiver, a sender cannot learn which data are chosen by the receiver, and a receiver cannot learn the data which were not chosen. In k-out-of-n OT, a receiver chooses k data from n data.

3. ONE-TIME PROGRAMS WITH CLOUD STORAGE

In this section, we propose an OTP scheme which does not require OTMs by utilizing multiple cloud servers.

3.1 OTP with n Servers

In an OTM, only one of two values can be read out. We realize this functionality by using multiple servers.

Suppose that three types of parties: OTP-generator $\mathsf{G_{OTP}}$, OTP-executor $\mathsf{E_{OTP}}$, cloud servers $\mathsf{S_j}$ $(1 \le j \le n)$ that conduct electronic transactions.

A transaction is performed in two steps:

Step1 - Generation step. In this step, $\mathsf{G_{OTP}}$ generates an OTP. We show the details of this step in Figure 1. $\mathsf{G_{OTP}}$ generates a GC and necessary information. Then $\mathsf{G_{OTP}}$ sends the necessary information to $\mathsf{E_{OTP}}$ and $\mathsf{S_j}$ $(1 \le j \le n)$.

Step1

1. $\mathsf{G_{OTP}}$ converts a logical circuit into a GC.
2. $\mathsf{G_{OTP}}$ generates necessary information:
   (a) $\mathsf{G_{OTP}}$ divides input labels $\{\ell_i^0,\ \ell_i^1\}_{1\le i\le m}$ for the GC into n shares $\{\ell_{i,j}^0,\ \ell_{i,j}^1\}_{1\le i\le m, 1\le j\le n}$ by using (k,n)-secret sharing.
   (b) $\mathsf{G_{OTP}}$ generates secret key sk by using a secret key generation algorithm.
   (c) $\mathsf{G_{OTP}}$ encrypts $\{\ell_{i,j}^0,\ \ell_{i,j}^1\}_{1\le i\le m, 1\le j\le n}$ by using sk.
   (d) $\mathsf{G_{OTP}}$ calculates $h(sk \parallel j)$ by using hash function $h(\cdot)$ as an index of the shares.
   Here $a \parallel b$ means a concatenation of bit strings of a and b.
3. $\mathsf{G_{OTP}}$ sends the necessary information:
   (a) $\mathsf{G_{OTP}}$ sends the GC and sk to $\mathsf{E_{OTP}}$.
   (b) $\mathsf{G_{OTP}}$ sends $\{E_{sk}(\ell_{i,j}^0),\ E_{sk}(\ell_{i,j}^1)\}_{1\le i\le m}$ and $h(sk \parallel j)$ to each $\mathsf{S_j}$ $(1 \le j \le n)$.

Figure 1: Generation step

Step2 - Execution step. In this step, $\mathsf{E_{OTP}}$ executes the OTP. We show the details of this step in Figure 2 . $\mathsf{E_{OTP}}$ performs a transaction with $\mathsf{S_j}$ $(1 \le j \le n)$ to obtain the labels corresponding to the inputs of the GC.

Step2

1. $\mathsf{E_{OTP}}$ performs a transaction with $\mathsf{S_j}$:
   (a) $\mathsf{E_{OTP}}$ calculates $h(sk \parallel j)$ as an index of his shares.
   (b) $\mathsf{E_{OTP}}$ sends $h(sk \parallel j)$ to $\mathsf{S_j}$.
   (c) $\mathsf{S_j}$ identifies the shares which $\mathsf{E_{OTP}}$ is retrieving by using $h(sk \parallel j)$.
   (d) $\mathsf{E_{OTP}}$ chooses input $b \in \{0,1\}^m$ for the OTP. Here b_i means the ith bit of b.
   (e) $\mathsf{E_{OTP}}$ obtains $\{E_{sk}(\ell_{i,j}^{b_i})\}_{1\le i\le m}$ by performing 1-out-of-2 OT with $\mathsf{S_j}$.
   (f) $\mathsf{E_{OTP}}$ decrypts $\{E_{sk}(\ell_{i,j}^{b_i})\}_{1\le i\le m}$ by using sk and obtains the shares of the necessary input labels.

[1] In [4], a tweaked GC is actually used to realize an OTP and the tweak is needed for the security proof. However, for simplicity, here we use a normal GC to describe our scheme because it does not affect the functionality of an OTP.

(g) $\mathsf{S_j}$ erases the ciphertexts of the labels related to the index $h(sk \parallel j)$.

2. $\mathsf{E_{OTP}}$ performs the above operation with k servers and obtains k shares.
3. $\mathsf{E_{OTP}}$ reconstructs the input labels by using the retrieved shares.
4. $\mathsf{E_{OTP}}$ executes the GC with $\{\ell_i^{b_i}\}_{1\le i\le m}$.

Figure 2: Execution step

Even if one does not use cloud servers, the similar functionality can be realized by $\mathsf{G_{OTP}}$ and $\mathsf{E_{OTP}}$ directly performing 1-out-of-2 OT. In that case, there is no need to perform encryption processing, secret sharing processing, and communication with the servers. On the other hand, the advantage of our scheme is that $\mathsf{G_{OTP}}$ needs to be involved only in the first process. Therefore $\mathsf{G_{OTP}}$ does not need to be online, so $\mathsf{E_{OTP}}$ can decide the input at an arbitrary timing, and $\mathsf{G_{OTP}}$ never knows its timing.

3.2 Security Analysis

Attack by Servers. Since 1-out-of-2 OT is used, the servers cannot know the input chosen by $\mathsf{E_{OTP}}$.

Even if the servers collaborate with each other, since it is not possible to generate an index $h(sk \parallel j)$ without knowing sk, the servers cannot learn which data belongs to the same $\mathsf{E_{OTP}}$.

Attack by $\mathsf{E_{OTP}}$. Suppose a malicious $\mathsf{E_{OTP}}$ tries to run a GC more than once with different inputs. In that case it is necessary to corrupt the $2k-n$ servers to obtain both the input labels (see the setting of k, n below). By assuming that it is difficult to corrupt $2k-n$ (or more) cloud servers, our scheme can be secure. Here the corruption of a server by an adversary means that the adversary can know all the information stored on the server, and it makes it possible for the adversary to obtain both the labels.

Attack by a Third party. If a server cannot identify the correct $\mathsf{E_{OTP}}$, it may send the shares to a invalid third party and then delete the shares without being retrieved by the correct $\mathsf{E_{OTP}}$. Here we can assume that having the index $h(sk \parallel j)$ is the proof that the request is from the correct $\mathsf{E_{OTP}}$. If taken from a sufficiently large space, we can ignore the probability that a third party who does not have sk by chance generates the correct index.

Setting of k, n. Now, assume using $(k, k+1)$-secret sharing. When $\mathsf{E_{OTP}}$ corrupted $k-1$ servers, if he requests the shares of label $b = 0^m$ from one of the two remaining servers and the shares of label $b = 1^m$ from the other, he can reconstruct the labels corresponding to all the inputs.

Similarly, when using the (k, n)-secret sharing, $\mathsf{E_{OTP}}$ can reconstruct the labels corresponding to all the inputs by corrupting $2k-n$ servers. Thus, in increasing n, the number of servers that $\mathsf{E_{OTP}}$ must corrupt to reconstruct all the labels corresponding to all the inputs is reduced. Therefore if we assume that a malicious $\mathsf{E_{OTP}}$ can corrupt at most k' servers, k and n need to be set such that $k' < 2k-n$ and we can set $k = n$ if we sacrifice fault tolerance.

The advantage of setting $k < n$ is that $\mathsf{E_{OTP}}$ can execute the OTP even if $n-k$ servers are offline, and data will be more resistant to loss and damage.

It is possible to set $k = n = 1$, i.e., only one cloud server exists. However, it is desirable that k be a somewhat large number, as $\mathsf{E_{OTP}}$ and the server can collude easily.

We also note that in order to prevent $\mathsf{E_{OTP}}$ from requesting the shares of labels twice, k and n need to be set such that $n-k < k$.

In order to solve the trade-off between the security and availability, for example, we can use a method like a reputation system keeping a list of trusted servers and removing a corrupted server when it goes wrong.

4. APPLICATION OF OTP TO ELECTRONIC MONEY

In this section, we consider an electronic money scheme as an application of the proposed OTP.

Suppose that there are three parties Bank, Shop, User that conduct electronic money transactions.

A transaction is performed in three steps. First, Bank issues electronic money in response to User's request, and User pays it to Shop, then Shop requests the realization of electronic money from Bank. We show an overview of the transaction in Figure 3.

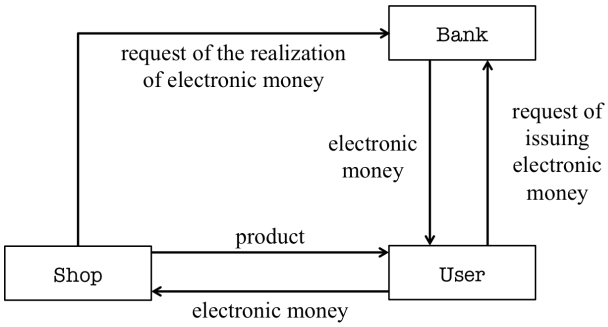

Figure 3: Transaction Overview

The proposed electronic money scheme (in Section 4.1) can prevent double-spending based on the security of OTPs. In order to cope with the case where the security of OTPs is broken, we propose another scheme (in Section 4.3) in which Bank can identify the ID of a malicious User at the time of realization of electronic money by Shop if User double-spends electronic money. The latter scheme is less efficient because of its additional robustness.

4.1 Electronic Money Scheme Preventing Double-Spending

We assume the following facts. Bank uses a digital signature scheme [1]. The digital signature scheme has a signing function Sign and verifying function Verify. Bank has a public key pk_{bank} and a secret key sk_{bank} for the digital signature scheme. Bank has a database DB to store the history of the realization. Shop and User each have an account at Bank.

A transaction is performed in three steps:

Step1 - Bank and User. In this step, Bank issues electronic money to User. We show the details of this step in Figure 4.

Step1

1. User requests electronic money for Y dollars from Bank.
2. Bank generates and sends $[\mathsf{Sign}(sk_{\mathrm{bank}},\ Y \parallel \cdot)]_{\mathsf{OTP}}$ to User.
   Here $[f(\cdot)]_{\mathsf{OTP}}$ means a function $f(\cdot)$ converted into the OTP, that is, a set of GC and sk.
3. Bank withdraws Y dollars from the account of User.

Figure 4: Bank **and** User

Step2 - User and Shop. In this step, User pays electronic money, Y dollar to Shop. We show the details of this step in Figure 5.

Step2

1. Shop generates a random number R and calculates $r \leftarrow h(R)$ with hash function $h(\cdot)$, and sends r to User.
2. User calculates the digital signature $z \leftarrow [\mathsf{Sign}(sk_{\mathrm{bank}},\ Y \parallel r)]_{\mathsf{OTP}}$ by executing the OTP and sends z to Shop.
3. If $\mathsf{Verify}(pk_{\mathrm{bank}},\ Y \parallel r,\ z) = \mathsf{true}$, the transaction is established.

Figure 5: User **and** Shop

Step3 - Shop and Bank. In this step, Shop requests the realization of electronic money from Bank. We show the details of this step in Figure 6.

Step3

1. Shop sends $(z,\ Y)$ to Bank.
2. If $(z,\ Y,\ R)$ is already stored in DB, Bank shows R to Shop and refuses the realization, and otherwise it requests R from Shop.
3. Shop sends R to Bank.
4. If $\mathsf{Verify}(pk_{\mathrm{bank}},\ Y \parallel h(R),\ z) = \mathsf{true}$, Bank performs the realization, i.e., it credits Y dollars to the account of Shop.
5. Bank stores $(z,\ Y,\ R)$ in DB as the spent electronic money.

Figure 6: Shop **and** Bank

If we set an expiration date for each electronic money by, for example, generating an OTP $[\mathsf{Sign}(sk_{\mathrm{bank}}, Y \parallel date \parallel \cdot]_{\mathsf{OTP}}$ and deleting items that expired from DB, it reduces the amount of DB although it also reduces the usability of electronic money.

4.2 Security Analysis

Anonymity. From the viewpoint of anonymity, Bank should not be able to trace electronic money of User.

In our scheme, we use an OTP that generates a digital signature. Therefore, by using pk_{bank}, any User can verify the validity of the generated signature and confirm that sk_{bank} is indeed included in the electronic money (i.e., the OTP). On the other hand, pk_{bank} is a public key common to the users. Therefore, given a signature, it is difficult to identify the OTP that generated the signature. Thus, Bank cannot trace User from the signature sent by Shop to Bank.

Attack by User. Even if User attempts to request the realization by sending $(z,\ Y)$ to Bank before Shop sends (z, Y) to Bank, when Bank requests R from him, User cannot calculate R from r, so the attack cannot be successful.

Double-spending by User. When User double-spends electronic money in existing electronic money schemes, that fact is not detected until Bank checks the database of spent electronic money. In our scheme, User needs to generate a digital signature for the random number sent by Shop at the time of transaction with Shop. Then User cannot sign two or more random numbers because of the functionality of OTPs, so User cannot perform double-spending.

However, a malicious User that executes a GC any number of times by corrupting $2k - n$ servers can perform double-spending. In Section 4.3, to cope with that case, we mention an approach that combines our scheme with an existing electronic money scheme so that we can detect such double-spending.

Attack by Shop. Even if Shop tries to request the realization from Bank by sending $(z,\ Y)$ twice, Bank can reject it because (z, Y) is already stored in DB.

Shop cannot generate z, and changing $Y,\ R$ does not pass the signature verification, so Shop cannot request the realization by itself.

Attack by Bank. Bank may try to deny the realization by saying that (z, Y, R) is already stored in DB, but Bank cannot deny the realization because Bank cannot show R based on the one-wayness of hash function $h(\cdot)$.

Attack by a Third party. After a legal Shop finishes the realization, if an attacker who stole (z, Y) somehow requests the realization from Bank, the attacker can obtain R from Bank, but because the transaction has been completed, there is no problem.

4.3 Electronic Money Scheme Preventing and Detecting Double-Spending

By combining our scheme with an existing scheme called offline electronic money [2], we can have both the advantages of our scheme and the existing scheme.

As a result, User who cannot break the security of OTPs cannot perform double-spending, and a more powerful User who breaks the securiy of OTPs (i.e., an adversary that can corrupt $2k-n$ (or more) cloud servers) can be detected as a

malicious User in the realization step of Bank just as in the existing scheme.

However, in the combined scheme, the efficiency is reduced because the number of necessary processes is increased.

We show below an example of our scheme combined with the existing method called cut-and-choose method [2].

In addition to the facts presented in Section 4.1, we assume the following facts. Bank has an RSA public key $(e,\ n)$ corresponding to Y and an RSA secret key d. User has a unique ID. A one-way function $f(\cdot)$ is known to all the parties.

A transaction is performed in three steps:

Step1 - Bank and User. In this step, Bank issues electronic money to User. We show the details of this step in Figure 7.

Step1

1. User generates random numbers $a_1, \cdots, a_k$ and s_i.
2. User calculates $(x_i,\ y_i) \leftarrow (h(a_i),\ h(a_i \oplus ID))$ and $A_i \leftarrow f(x_i,\ y_i)s_i^e \bmod n$ for $1 \leq i \leq k$.
3. User sends $(A_1, \cdots, A_k)$ to Bank.
4. Bank divides $(1, \cdots, k)$ into two subsets $(t_1, \cdots, t_{k/2})$, $(t_{k/2+1}, \cdots, t_k)$ at random.
5. Bank sends $(t_{k/2+1}, \cdots, t_k)$ to User.
6. User sends $(a_{t_{k/2+1}}, \cdots, a_{t_k})$, $(s_{t_{k/2+1}}, \cdots, s_{t_k})$ to Bank.
7. Bank confirms that User generated A_i properly by calculating each data.
8. Bank calculates $D \leftarrow (A_{t_1} \times \cdots \times A_{t_{k/2}})^d \bmod n$, and sends it to User.
9. Bank sends $[\mathsf{Sign}(sk_{\mathrm{bank}},\ Y \parallel \cdot)]_{\mathtt{OTP}}$ to User.
10. User calculates $C \leftarrow D/(s_{t_1} \times \cdots \times s_{t_{k/2}}) = (f(x_{t_1},\ y_{t_1}) \times \cdots \times f(x_{t_{k/2}},\ y_{t_{k/2}}))^d \bmod n$.
11. Bank withdraws Y dollars from the account of User.

Figure 7: Bank **and** User

Step2 - User and Shop. In this step, User pays electronic money, Y dollar to Shop. We show the details of this step in Figure 8.

Step2

1. User sends C to Shop.
2. Shop generates a random number R and calculates $r \leftarrow h(R)$.
   Here let $(r_1, \cdots, r_{k/2})$ be the first $k/2$ bit of r.
3. Shop sends r to User.
4. For each $j = 1, \cdots, k/2$, if $r_j = 0$ then User calculates $U_j \leftarrow (a_{t_j},\ y_{t_j})$, if $r_j = 1$ then User calculates $U_j \leftarrow (x_{t_j},\ a_{t_j} \oplus ID)$.
5. User calculates $z \leftarrow [\mathsf{Sign}(sk_{\mathrm{bank}},\ Y \parallel r)]_{\mathtt{OTP}}$.
6. User sends $U = (U_1, \cdots, U_{k/2})$ and z to Shop.
7. Shop calculates (x_{t_i}, y_{t_i}) from $U, R, h(\cdot)$.
8. For each j, Shop calculates $f(x_{t_j},\ y_{t_j})$, and confirms that $C^e = \prod_{j=1}^{k/2} f(x_{t_j},\ y_{t_j}) \bmod n$.
9. If $\mathsf{Verify}(pk_{\mathrm{bank}},\ Y \parallel r,\ z) = \mathsf{true}$, the transaction is established.

Figure 8: User **and** Shop

Step3 - Shop and Bank. In this step, Shop requests the realization of electronic money from Bank. We show the details of this step in Figure 9.

Step3

1. Shop sends $(C,\ U,\ z)$ to Bank.
2. If $(C,\ U,\ R,\ z)$ is already stored in DB, Bank shows R to Shop and refuses the realization, and otherwise it requests R from Shop.
3. Shop sends R to Bank.
4. If $\mathsf{Verify}(pk_{\mathrm{bank}},\ Y \parallel h(R),\ z) = \mathsf{true}$, Bank confirms validity of C by verifying U, and Bank performs the realization.
5. Bank stores $(C,\ U,\ R,\ z)$ in DB as the spent electronic money.

Figure 9: Shop **and** Bank

4.4 Security Analysis

Anonymity. Since C is the blind signature, Bank cannot learn C and $f(x_i,\ y_i)$. Therefore Bank cannnot trace User.

Double-spending by User. When a malicious User uses the same money twice, Bank can identify his ID with high possibility by checking the electronic money against C stored in DB and calculating $ID \leftarrow (a_i) \oplus (a_i \oplus ID)$.

Therefore, in this scheme, Bank can detect ID of the double-spender even if he can corrupt $2k - n$ servers for OTPs.

5. CONCLUSION

We proposed an OTP which does not require special hardware by distributing the encrypted shares of labels to multiple cloud servers using (k, n)-secret sharing and encryption of shares. In addition, we proposed the electronic money scheme that prevents double-spending by allowing User to generate a digital signature necessary for a transaction with Shop only once by using an OTP. By combining our scheme

with an existing offline scheme, we can detect double-spending even by an adversary who can generate digital signatures multiple times although the efficiency is reduced instead.

6. ACKNOWLEGDEMENTS

This work was supported in part by JSPS KAKENHI Grant Number 26330151 and JSPS and DST under the Japan - India Science Cooperative Program.

7. REFERENCES

[1] M. Bellare and P. Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 399–416. Springer, 1996.

[2] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Proceedings on Advances in cryptology*, pages 319–327. Springer-Verlag New York, Inc., 1990.

[3] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[4] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. One-time programs. In *Annual International Cryptology Conference*, pages 39–56. Springer, 2008.

[5] M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[6] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[7] P. Snyder. Yao 's garbled circuits: Recent directions and implementations. 2014.

[8] A. C.-C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.