This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2520886, IEEE Transactions on Information Forensics and Security

Identity-Based Proxy-Oriented Data Uploading and Remote Data Integrity Checking in Public Cloud

Huaqun Wang, Debiao He, Shaohua Tang

***Abstract*—More and more clients would like to store their data to PCS (public cloud servers) along with the rapid development of cloud computing. New security problems have to be solved in order to help more clients process their data in public cloud. When the client is restricted to access PCS, he will delegate its proxy to process his data and upload them. On the other hand, remote data integrity checking is also an important security problem in public cloud storage. It makes the clients check whether their outsourced data is kept intact without downloading the whole data. From the security problems, we propose a novel proxy-oriented data uploading and remote data integrity checking model in identity-based public key cryptography: ID-PUIC (identity-based proxy-oriented data uploading and remote data integrity checking in public cloud). We give the formal definition, system model and security model. Then, a concrete ID-PUIC protocol is designed by using the bilinear pairings. The proposed ID-PUIC protocol is provably secure based on the hardness of CDH (computational Diffie-Hellman) problem. Our ID-PUIC protocol is also efficient and flexible. Based on the original client's authorization, the proposed ID-PUIC protocol can realize private remote data integrity checking, delegated remote data integrity checking and public remote data integrity checking.**

***Index Terms*—Cloud computing, Identity-based cryptography, Proxy public key cryptography, Remote data integrity checking**

I. Introduction

Along with the rapid development of computing and communication technique, a great deal of data are generated. These massive data needs more strong computation resource and greater storage space. Over the last years, cloud computing satisfies the application requirements and grows very quickly. Essentially, it takes the data processing as a service, such as storage, computing, data security, *etc*. By using the public cloud platform, the clients are relieved of the burden for storage management, universal data access with independent geographical locations, *etc*. Thus, more and more clients would like to store and process their data by using the remote cloud computing system.

In public cloud computing, the clients store their massive data in the remote public cloud servers. Since the stored data is outside of the control of the clients, it entails the security risks in terms of confidentiality, integrity and availability of data and service. Remote data integrity checking is a primitive which can be used to convince the cloud clients that their data are kept intact. In some special cases, the data owner may be restricted to access the public cloud server, the data owner will delegate the task of data processing and uploading to the third party, for example the proxy. On the other side, the remote data integrity checking protocol must be efficient in order to make it suitable for capacity-limited end devices. Thus, based on identity-based public cryptography and proxy public key cryptography, we will study ID-PUIC protocol .

A. Motivation

In public cloud environment, most clients upload their data to *PCS* and check their remote data's integrity by Internet. When the client is an individual manager, some practical problems will happen. If the manager is suspected of being involved into the commercial fraud, he will be taken away by the police. During the period of investigation, the manager will be restricted to access the network in order to guard against collusion. But, the manager's legal business will go on during the the period of investigation. When a large of data is generated, who can help him process these data ? If these data cannot be processed just in time, the manager will face the lose of economic interest. In order to prevent the case happening, the manager has to delegate the proxy to process its data, for example, his secretary. But, the manager will not hope others have the ability to perform the remote data integrity checking. Public checking will incur some danger of leaking the privacy. For example, the stored data volume can be detected by the malicious verifiers. When the uploaded data volume is confidential, private remote data integrity checking is necessary. Although the secretary has the ability to process and upload the data for the manager, he still cannot check the manager's remote data integrity unless he is delegated by the manager. We call the secretary as the proxy of the manager.

In PKI (public key infrastructure), remote data integrity checking protocol will perform the certificate management. When the manager delegates some entities to perform the remote data integrity checking, it will incur considerable overheads since the verifier will check the certificate when it checks the remote data integrity. In PKI, the considerable overheads come from the heavy certificate verification, certificates generation, delivery, revocation, renewals, *etc*. In public cloud computing, the end devices may have low computation capacity, such as mobile phone, ipad, *etc*. Identity-based public key cryptography can eliminate the complicated certificate

H. Wang is with the College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China, and also with the State Key Laboratory of Cryptology, Beijing 100878, China (e-mail: wanghuaqun@aliyun.com).

D. He is with the State Key Laboratory of Software Engineering, Computer School, Wuhan University, Wuhan 430072, China (e-mail: hedebiao@163.com).

S. Tang is with the School of Computer Science, South China University of Technology, Guangzhou 510006, China (e-mail: shtang@ieee.org).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2520886, IEEE Transactions on Information Forensics and Security

management. In order to increase the efficiency, identity-based proxy-oriented data uploading and remote data integrity checking is more attractive. Thus, it will be very necessary to study the ID-PUIC protocol.

B. Related work

There exist many different security problems in the cloud computing [1], [2]. This paper is based on the research results of proxy cryptography, identity-based public key cryptography and remote data integrity checking in public cloud. In some cases, the cryptographic operation will be delegated to the third party, for example proxy. Thus, we have to use the proxy cryptography. Proxy cryptography is a very important cryptography primitive. In 1996, Mambo *et al.* proposed the notion of the proxy cryptosystem [3]. When the bilinear pairings are brought into the identity-based cryptography, identity-based cryptography becomes efficient and practical. Since identity-based cryptography becomes more efficient because it avoids of the certificate management, more and more experts are apt to study identity-based proxy cryptography. In 2013, Yoon *et al.* proposed an ID-based proxy signature scheme with message recovery [4]. Chen *et al.* proposed a proxy signature scheme and a threshold proxy signature scheme from the Weil pairing [5]. By combining the proxy cryptography with encryption technique, some proxy re-encryption schemes are proposed. Liu *et al.* formalize and construct the attribute-based proxy signature [6]. Guo *et al.* presented a non-interactive CPA(chosen-plaintext attack)-secure proxy re-encryption scheme, which is resistant to collusion attacks in forging re-encryption keys [7]. Many other concrete proxy re-encryption schemes and their applications are also proposed [8], [9], [10].

In public cloud, remote data integrity checking is an important security problem. Since the clients' massive data is outside of their control, the clients' data may be corrupted by the malicious cloud server regardless of intentionally or unintentionally. In order to address the novel security problem, some efficient models are presented. In 2007, Ateniese *et al.* proposed provable data possession (PDP) paradigm [11]. In PDP model, the checker can check the remote data integrity without retrieving or downloading the whole data. PDP is a probabilistic proof of remote data integrity checking by sampling random set of blocks from the public cloud server, which drastically reduces I/O costs. The checker can perform the remote data integrity checking by maintaining small metadata. After that, some dynamic PDP model and protocols are designed [12], [13], [14], [15], [16]. Following Ateniese *et al.*'s pioneering work, many remote data integrity checking models and protocols have been proposed [17], [18], [19]. In 2008, proof of retrievability (POR) scheme was proposed by Shacham *et al.* [20]. POR is a stronger model which makes the checker not only check the remote data integrity but also retrieve the remote data. Many POR schemes have been proposed [21], [22], [23], [24], [25], [26]. On some cases, the client may delegate the remote data integrity checking task to the third party. In cloud computing, the third party auditing is indispensable [27], [28], [29], [30]. By using cloud storage, the clients can access the remote data with independent geographical locations. The end devices may be mobile and limited in computation and storage. Thus, efficient and secure ID-PUIC protocol is more suitable for cloud clients equipped with mobile end devices.

From the role of the remote data integrity checker, all the remote data integrity checking protocols are classified into two categories: private remote data integrity checking and public remote data integrity checking. In the response checking phase of private remote data integrity checking, some private information is indispensable. On the contrary, private information is not required in the response checking of public remote data integrity checking. Specially, when the private information is delegated to the third party, the third party can also perform the remote data integrity checking. In this case, it is also called delegated checking.

C. Contributions

In public cloud, this paper focuses on the identity-based proxy-oriented data uploading and remote data integrity checking. By using identity-based public key cryptology, our proposed ID-PUIC protocol is efficient since the certificate management is eliminated. ID-PUIC is a novel proxy-oriented data uploading and remote data integrity checking model in public cloud. We give the formal system model and security model for ID-PUIC protocol. Then, based on the bilinear pairings, we designed the first concrete ID-PUIC protocol. In the random oracle model, our designed ID-PUIC protocol is provably secure. Based on the original client's authorization, our protocol can realize private checking, delegated checking and public checking.

D. Paper Organization

The paper is organized below. The formal system model and security model of ID-PUIC protocol are given in Section II. The concrete protocol, performance analysis and prototype implementation are presented in Section III. Section IV analyzes the proposed ID-PUIC protocol's security. The proposed protocol is provably secure. At the end of the paper, the conclusion is given in Section V.

II. System model and security model of ID-PUIC

In this section, we give the system model and security model of ID-PUIC protocol. An ID-PUIC protocol consists of four different entities which are described below:

1) *OriginalClient*: an entity, which has massive data to be uploaded to *PCS* by the delegated proxy, can perform the remote data integrity checking.
2) *PCS* (Public Cloud Server): an entity, which is managed by cloud service provider, has significant storage space and computation resource to maintain the clients' data.
3) *Proxy*: an entity, which is authorized to process the *OriginalClient*'s data and upload them, is selected and authorized by *OriginalClient*. When *Proxy* satisfies the warrant m_ω which is signed and issued by *OriginalClient*, it can process and upload the original client's data; otherwise, it can not perform the procedure.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2520886, IEEE Transactions on Information Forensics and Security

4) *KGC* (Key Generation Center): an entity, when receiving an identity, it generates the private key which corresponds to the received identity.

In our proposed ID-PUIC protocol, *OriginalClient* will interact with *PCS* to check the remote data integrity. Thus, we give the the definition of interactive proof system. Then, we give the formal definition and security model of ID-PUIC protocol.

Definition 1 (Interactive proof system): [31] Let $c, s : \mathbb{N} \to \mathbb{R}$ be functions satisfying $c(n) > s(n) + \frac{1}{p(n)}$ for some polynomial $p(\cdot)$. An interactive pair (P, V) is called an interactive proof system for the language L, with completeness bound $c(\cdot)$ and soundness bound $s(\cdot)$, if

1) Completeness: for every $x \in L$, $\Pr[< P, V > (x) = 1] \geq c(|x|)$.
2) Soundness: for every $x \notin L$ and every interactive machine B, $\Pr[< B, V > (x) = 1] \leq s(|x|)$.

In the definition of ID-PUIC, *i.e.*, Definition 2, we will take use of the interactive proof system.

Definition 2 (ID-PUIC): An ID-PUIC protocol is a collection of four phases (*Setup*, *Extract*, *Proxy-key Generation*, *TagGen*) and an interactive proof system (*Proof*). The detailed phases are described below.

1) *Setup*: When the security parameter *k* is input, the algorithm outputs the system public parameters and the master secret key. The system public parameters are made public and the master secret key *msk* is made confidential by KGC.
2) *Extract*: When the system public parameters, the master secret key *msk*, and an identity *ID* are input, KGC outputs the private key sk_{ID} which corresponds to the identity *ID*.
3) *Proxy-key Generation*: *OriginalClient* generates the warrant m_ω and signs m_ω. Then, it sends the warrant-signature pair to the proxy. Upon receiving the warrant-signature pair from *OriginalClient*, the proxy generates the proxy-key by using its own private key.
4) *TagGen*: Input the file block F_i and the proxy-key, the proxy generates the corresponding tag T_i. Then, it uploads the block-tag pair to *PCS*.
5) *Proof*: It is an interactive proof system between *PCS* and *OriginalClient*. At the end of the interactive proof protocol, *OriginalClient* outputs a bit $\{0|1\}$ denoting "success" or "failure".

A practical ID-PUIC protocol must be efficient and provably secure. Based on the communication and computation overheads, efficiency analysis can be given. On the other hand, a secure ID-PUIC protocol must satisfy the following security requirements:

1) *OriginalClient* can perform the ID-PUIC protocol without the local copy of the file(s) to be checked.
2) Only if the proxy is authorized, *i.e.*, it satisfies the warrant m_ω, the proxy can process the files and upload the block-tag pairs on behalf of *OriginalClient*.
3) *OriginalClient* cannot counterfeit the proxy to generate block-tag pairs, *i.e.*, the proxy-protection property is satisfied.
4) If some challenged block-tag pairs are modified or lost, *PCS*'s response cannot pass *OriginalClient*'s integrity checking.

To capture the above security requirements, we formalize the security definition of an ID-PUIC protocol. First, we give the formal definition of proxy-protection.

Definition 3 (Proxy-protection): An ID-PUIC protocol satisfies the property of proxy-protection if for the probabilistic polynomial time adversary $\mathcal{A}_1$, the probability that $\mathcal{A}_1$ wins the ID-PUIC game-1 is negligible. The ID-PUIC game-1 between $\mathcal{A}_1$ and the challenger $\mathcal{C}_1$ is given below:

1) *Setup*: The challenger $\mathcal{C}_1$ runs $Setup$ and gets the system public parameters and master secret key. By running *Extract*, $\mathcal{C}_1$ gets *OriginalClient* ID_o's private key sk_{ID_o} and the proxy ID_p's private key sk_{ID_p}. It sends the public parameters and *OriginalClient* ID_o's private key sk_{ID_o} to $\mathcal{A}_1$ while it keeps confidential the master secret key msk and the proxy ID_p's private key sk_{ID_p}.
2) *Oracle queries*: $\mathcal{A}_1$ adaptively queries the oracles *Extract*, *Hash*, *Proxy-key Generation*, *TagGen* to $\mathcal{C}_1$ below:
   - *Extract* queries. $\mathcal{A}_1$ queries the entity *ID*'s private key to $\mathcal{C}_1$. For the identity *ID*, $\mathcal{C}_1$ runs *Extract* and gets the private key sk_{ID}. Then, it forwards sk_{ID} to $\mathcal{A}_1$. Denote S as the queried identity set in the phase. The restriction is that ID_p cannot be queried in the phase, *i.e.*, $ID_p \notin S$.
   - *Hash* queries. $\mathcal{A}_1$ queries *hash* oracle to $\mathcal{C}_1$ adaptively. $\mathcal{C}_1$ responds $\mathcal{A}_1$ the *hash* values.
   - *Proxy-key Generation* queries. $\mathcal{A}_1$ sends (ID'_o, ID'_p) to $\mathcal{C}_1$ and queries the proxy-key where the original client is ID'_o and the proxy is ID'_p. Denote S' as the set which is composed of all the queried original client identity and proxy identity pairs. The restriction is that $(ID_o, ID_p) \notin S'$.
   - *TagGen* queries. $\mathcal{A}_1$ makes block-tag pair queries adaptively. For the block F_i, $\mathcal{C}_1$ computes its tag T_i and responds $\mathcal{A}_1$ with T_i.

   At the end of game-1, $\mathcal{A}_1$ outputs the forged block-tag pair (F', T') with non-negligible probability, where F' has not been queried to *TagGen* oracle. If (F', T') is valid block-tag pair, then $\mathcal{A}_1$ wins the above game with non-negligible probability.

The security definition 3 gives the formal security definition for proxy-protection property of ID-PUIC protocol. Let *OriginalClient* be the adversary. If *OriginalClient* cannot win the ID-PUIC game-1 with non-negligible probability, then the ID-PUIC protocol satisfies the proxy-protection property. The following definition formalizes another security property of ID-PUIC protocol, *i.e.*, *Unforgeability*.

Definition 4 (Unforgeability): An ID-PUIC protocol satisfies the security property of unforgeability if for any probabilistic polynomial time adversary $\mathcal{A}_2$ (*i.e.*, malicious PCS) the probability that $\mathcal{A}_2$ wins the following ID-PUIC game-2 on a set of file blocks is negligible. Let ID_o be the original client. Let ID_p be ID_o's proxy who uploads the block-tag pairs to *PCS*. The ID-PUIC game-2 between the adversary $\mathcal{A}_2$ and the challenger $\mathcal{C}_2$ is given below:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2520886, IEEE Transactions on Information Forensics and Security

1) *Setup*: $\mathcal{C}_2$ runs $Setup$ and gets system public parameters and master secret key. It sends the system public parameters to $\mathcal{A}_2$ while it keeps confidential the master secret key msk.
2) *First-Phase Queries*: $\mathcal{A}_2$ adaptively makes *Extract, Hash, Proxy-key Generation, TagGen* queries to $\mathcal{C}_2$. $\mathcal{C}_2$ responds $\mathcal{A}_2$.
   - *Extract* queries. $\mathcal{A}_2$ queries *ID*'s private key. By running *Extract*, $\mathcal{C}_2$ gets the corresponding private key sk_{ID} and sends it to $\mathcal{A}_2$. Denote S_1 as the queried identity set in the first-phase. The restriction is that $ID_p \notin S_1$.
   - *Hash* queries. $\mathcal{A}_2$ queries *Hash* oracle adaptively. $\mathcal{C}_2$ sends *Hash* values to $\mathcal{A}_2$.
   - *Proxy-key Generation* queries. $\mathcal{A}_2$ sends (ID'_o, ID'_p) to $\mathcal{C}_2$ and queries the proxy-key where the original client is ID'_o and the proxy is ID'_p. Denote S_2 as the set which is composed of all the queried original client identity and proxy identity pairs. The restriction is that $(ID_o, ID_p) \notin S_2$.
   - *TagGen* queries. $\mathcal{A}_2$ makes block-tag pair queries adaptively. For the block F_i, $\mathcal{C}_2$ computes the tag T_i and sends it back to $\mathcal{A}_2$. Denote S_3 as the set of index that the corresponding block tags have been queried in the first-phase.
3) *Challenge*: For the original client ID_o and the proxy ID_p, $\mathcal{C}_2$ generates a challenge $chal$ which defines an ordered collection $\{i_1, i_2, \cdots, i_c\}$, where $ID_p \notin S_1, (ID_o, ID_p) \notin S_2$, $\{i_1, i_2, \cdots, i_c\} \nsubseteq S_3$, and c is a positive integer. $\mathcal{A}_2$ is required to provide the remote data integrity proof for the blocks $F_{i_1}, \cdots, F_{i_c}$.
4) Second-Phase Queries: Similar to the First-Phase Queries. Denote S'_1 as the queried identity set in *Extract* of the second-phase queries. Denote S'_2 as the set which is composed of all the queried original client identity and proxy identity pairs in *Proxy-key Generation* of the second-phase queries. Denote S'_3 as the set of index that the corresponding block tags have been queried in *TagGen* of the second-phase queries. The restriction is that $ID_p \notin S_1 \bigcup S'_1, (ID_o, ID_p) \notin S_2 \bigcup S'_2, \{i_1, i_2, \cdots, i_c\} \nsubseteq S'_3$.
5) Forge: $\mathcal{A}_2$ responses θ for the challenge $chal$.

If the response θ can pass $\mathcal{C}_2$'s verification, the adversary $\mathcal{A}_2$ wins the ID-PUIC game-2.

Definition 4 states that, if some challenged blocks have been modified or deleted, the malicious *PCS* cannot generate a valid remote data integrity proof. On the other hand, a practical ID-PUIC protocol also needs to convince the client that all of his outsourced data is kept integer with a high probability. The following definition states the security requirement.

Definition 5 ((ρ, δ) security): An ID-PUIC protocol satisfies the security (ρ, δ) if *PCS* corrupted ρ fraction of the whole blocks, the probability that the corrupted blocks are detected is at least δ.

The notations throughout this paper are listed in Table I.

TABLE I
NOTATIONS AND DESCRIPTIONS

Notations	Descriptions
$\mathcal{G}_1$	Cyclic multiplicative group with order q
$\mathcal{G}_2$	Cyclic multiplicative group with order q
$\mathcal{Z}_q^*$	$\{1, 2, \cdots, q-1\}$
g	A generator of $\mathcal{G}_1$
H, h	Two cryptographic hash functions
f	Pseudo-random function
π	Pseudo-random permutation
(x, Y)	Master secret/public key pair
(ID_o, sk_{ID_o})	Original client's identity-private key pair where $sk_{ID_o} = (R_o, \sigma_o)$
(ID_p, sk_{ID_p})	Proxy's identity-private key pair where $sk_{ID_p} = (R_p, \sigma_p)$
σ	The generated proxy-key
n	The block number
$F = (F'_1, \cdots, F'_n)$	The stored file F is split into n blocks
PCS	Public cloud server
O	Original client
u	The public parameter which is picked by the original client
(N_i, i)	N_i denotes the *i*-th block F'_i's name and other properties
(F_i, T_i)	The i-th block-tag pair
v_i	The permutated index $v_i = \pi_{k_1}(i)$ of i
$\theta = (\hat{F}, T)$,	PCS's response
C_p,	Time cost of bilinear pairings on $\mathcal{G}_1$
C_e,	Time cost of exponentiation on $\mathcal{G}_1$
C_m,	Time cost of multiplication on $\mathcal{G}_1$
m_ω,	The warrant which is generated and signed by the original client

III. THE PROPOSED ID-PUIC PROTOCOL

In this section, we propose an efficient ID-PUIC protocol for secure data uploading and storage service in public clouds. Bilinear pairings technique makes identity-based cryptography practical. Our protocol is built on the bilinear pairings. We first review the bilinear pairings. Then, the concrete ID-PUIC protocol is designed from the bilinear pairings. At last, based on the computation cost and communication cost, we give the performance analysis from two aspects: theoretical analysis and prototype implementation.

A. Bilinear pairings

Denote $\mathcal{G}_1$ and $\mathcal{G}_2$ as two cyclic multiplicative groups who have the same prime order q. Let $\mathcal{Z}_q^*$ denote the multiplicative group of the field $\mathcal{F}_q$. Bilinear pairings is a bilinear map $e : \mathcal{G}_1 \times \mathcal{G}_1 \to \mathcal{G}_2$ [32] which satisfies the properties below:

1) Bilinearity: $\forall g_1, g_2, g_3 \in \mathcal{G}_1$ and $a, b \in \mathcal{Z}_q^*$,
$$e(g_1, g_2 g_3) = e(g_2 g_3, g_1) = e(g_2, g_1) e(g_3, g_1)$$
$$e({g_1}^a, {g_2}^b) = e(g_1, g_2)^{ab}$$
2) Non-degeneracy: $\exists g_4, g_5 \in \mathcal{G}_1$ such that $e(g_4, g_5) \neq 1_{\mathcal{G}_2}$.
3) Computability: $\forall g_6, g_7 \in \mathcal{G}_1$, there is an efficient algorithm to compute $e(g_6, g_7)$.

The concrete bilinear pairings e can be constructed by using the modified Weil [33] or Tate pairings [34] on elliptic curves. Our ID-PUIC protocol construction takes use of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2520886, IEEE Transactions on Information Forensics and Security

easiness of DDH (Decisional Diffie-Hellman) problem while its security is based on the hardness of CDH (Computational Diffie-Hellman) problem. Let g be a generator of $\mathcal{G}_1$. CDH problem and DDH problem are defined below.

Definition 6 (CDH Problem on $\mathcal{G}_1$): Given the triple $(g, g^a, g^b) \in \mathcal{G}_1^3$ where $a, b \in \mathcal{Z}_q$ are unknown, compute $g^{ab} \in \mathcal{G}_1$.

Definition 7 (DDH Problem on $\mathcal{G}_1$): Given the quadruple $(g, g^a, g^b, \hat{g}) \in \mathcal{G}_1^4$ where $a, b \in \mathcal{Z}_q^*$ are unknown, decide whether or not the formula $g^{ab} = \hat{g}$ holds.

In the paper, we choose the group $\mathcal{G}_1$ which satisfies the condition that CDH problem is difficult but DDH problem is easy. On the group $\mathcal{G}_1$, DDH problem is easy by using the bilinear pairings. $(\mathcal{G}_1, \mathcal{G}_2)$ are also called GDH (Gap Diffie-Hellman) groups. On the groups $\mathcal{G}_1$ and $\mathcal{G}_2$, the basic requirement is that the DLP (Discrete Logarithm Problem) is difficult. Let g_2 be a generator of $\mathcal{G}_2$. It is given below:

Definition 8 (DLP on $\mathcal{G}_1$): Given (g, g^a) where $a \in \mathcal{Z}_q^*$ is unknown, it is difficult to calculate a.

Definition 9 (DLP on $\mathcal{G}_2$): Given (g_2, g_2^a) where $a \in \mathcal{Z}_q^*$ is unknown, it is difficult to calculate a.

In the paper, we choose the groups $\mathcal{G}_1$ and $\mathcal{G}_2$ which satisfy that DLP, CDH are difficult while DDH is easy.

B. Our Concrete ID-PUIC Protocol

This concrete ID-PUIC protocol comprises four procedures: *Setup*, *Extract*, *Proxy-key generation*, *TagGen*, and *Proof*. In order to show the intuition of our construction, the concrete protocol's architecture is depicted in Figure 1. First, *Setup* is performed and the system parameters are generated. Based on the generated system parameters, the other procedures are performed as Figure 1. It is described below: (1) In the phase *Extract*, when the entity's identity is input, KGC generates the entity's private key. Especially, it can generate the private keys for the client and the proxy. (2) In the phase *Proxy-key generation*, the original client creates the warrant and helps the proxy generate the proxy key. (3) In the phase *TagGen*, when the data block is input, the proxy generates the block's tag and upload block-tag pairs to *PCS*. (4) In the phase *Proof*, the original client *O* interacts with *PCS* . Through the interaction, *O* checks its remote data integrity.

Following the protocol's architecture, we give the concrete construction below. Without loss of generality, suppose that the proxy plans to upload the file F. According to the size of F, we split it into multiple blocks. Suppose that F is split into n blocks, *i.e.*, $F = (F_1, \cdots, F_n)$. F_i denotes the *i*-th block of F. Let N_i contain the name and attributes of the block F_i. (N_i, i) will be used to create the tag of F_i. The phases are described in detail as the following.

- *Setup*: Let $\mathcal{G}_1, \mathcal{G}_2$ be the two groups and *e* be the bilinear pairings which are given in the section III-A. Both $\mathcal{G}_1$ and $\mathcal{G}_2$ have the same order *q*. Let g be a generator g of the group $\mathcal{G}_1$. Two cryptographic hash functions are given below:

$$H : \{0,1\}^* \to \mathcal{Z}_q^*, \ h : \mathcal{Z}_q^* \times \{0,1\}^* \to \mathcal{G}_1$$

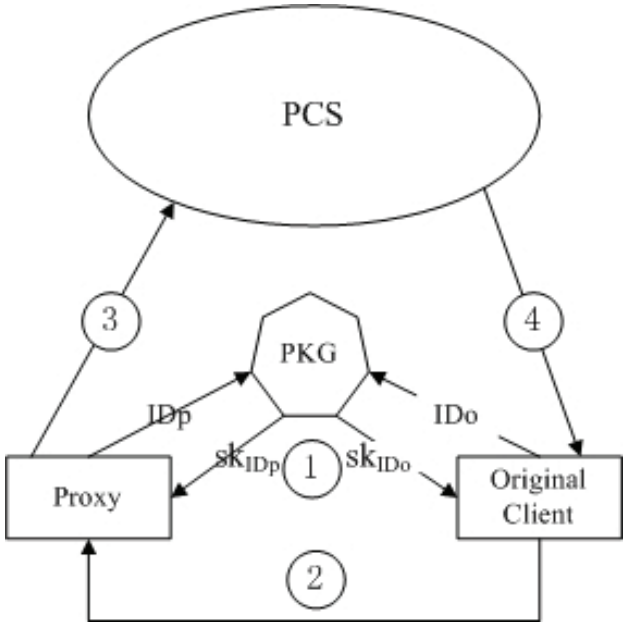


Fig. 1. Architecture of our ID-DPDP protocol

Pick a pseudo-random function f and a pseudo-random permutation π. The two functions f and π are defined below:

$$f : \mathcal{Z}_q^* \times \{1, 2, \cdots, n\} \to \mathcal{Z}_q^*$$
$$\pi : \mathcal{Z}_q^* \times \{1, 2, \cdots, n\} \to \{1, 2, \cdots, n\}$$

KGC generates its master secret key x where $x \in \mathcal{Z}_q^*$. Then, it computes $Y = g^x$. The parameters $\{\mathcal{G}_1, \mathcal{G}_2, e, q, g, Y, H, h, f, \pi\}$ are made public. The master secret key x is kept confidential by KGC.

- *Extract*: Input the original client's identity ID_o, KGC picks a random $r_o \in \mathcal{Z}_q^*$ and computes (R_o, σ_o) below:

$$R_o = g^{r_o}, \quad \sigma_o = r_o + xH(ID_o, R_o) \bmod q$$

Then, KGC sends $sk_{ID_o} = (R_o, \sigma_o)$ to the original client by the secure channel. Let sk_{ID_o} be the original client's private key. The original client verifies sk_{ID_o}'s correctness by verifying the following equation.

$$g^{\sigma_o} = R_o Y^{H(ID_o, R_o)} \tag{1}$$

If the formula (1) holds, the original client ID_o accepts sk_{ID_o} as its private key; otherwise, ID_o rejects it and requests its private key by using *Extract* again.
Similarly, input the proxy's identity ID_p, the proxy ID_p can also get its private key $sk_{ID_p} = (R_p, \sigma_p)$.

- *Proxy-key generation*: In order to generate the proxy key, the original client ID_o will interact with the proxy ID_p below:
  1) ID_o creates the warrant m_ω according to its requirements. The proxy ID_p cannot process and upload the original client ID_o's data unless it satisfies m_ω. ID_o picks a random $r_1 \in \mathcal{Z}_q^*$ and computes m_ω's signature below:

$$R_1 = g^{r_1}, \ \sigma_1 = r_1 + \sigma_o H(m_\omega, R_1) \bmod q$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2520886, IEEE Transactions on Information Forensics and Security

ID_o sends the warrant-signature pair $(m_\omega, R_1, \sigma_1)$ and R_o to ID_p and *PCS*.

2) ID_p checks the validity of $(m_\omega, R_1, \sigma_1, R_o)$ by verifying whether or not the following equation holds.

$$g^{\sigma_1} = R_1(R_o Y^{H(ID_o, R_o)})^{H(m_\omega, R_1)}$$

If the verification is unsuccessful, the proxy rejects it and informs ID_o; otherwise, it computes the proxy secret key:

$$\sigma = \sigma_1 + \sigma_p H(m_\omega, R_1)$$

The proxy secret key σ is kept secret by the proxy. At the same time, ID_p sends R_p to ID_o.

- *TagGen*: When ID_p satisfies the warrant m_ω, ID_p will help ID_o process its data. Suppose the original client's plaintext file is $\hat{F}$. By using the light-weight symmetric encryption, $\hat{F}$ is encrypted into the ciphertext F which will be uploaded to *PCS*. Based on the size of *F*, the proxy ID_p splits *F* into n blocks, *i.e.*, $F = (F_1, \cdots, F_n)$. F_i denotes the *i*-th block of F and $F_i \in \mathcal{Z}_q^*$. N_i contains the *i*-th block F_i's name and its properties. The proxy calculates $u = h(n+1, ID_0)$. Then, for $1 \le i \le n$, the proxy performs the following procedures step by step:
  1) The proxy computes $T_i = (h(i, N_i)u^{F_i})^\sigma$ by using the proxy key σ;
  2) The proxy outputs the block F_i's tag T_i.

  At last, the proxy gets all the block-tag pairs $\{(F_i, T_i), 1 \le i \le n\}$ and uploads them to *PCS*. When *PCS* receives m_ω's signature $(m_\omega, R_1, \sigma_1)$ and R_o, it checks $(m_\omega, R_1, \sigma_1)$'s validity by verifying whether $g^{\sigma_1} = R_1(R_o Y^{H(ID_o, R_o)})^{H(m_\omega, R_1)}$ holds. If it holds, *PCS* accepts m_ω; otherwise, it informs ID_o. When receiving the block-tag pairs $\{(F_i, T_i), 1 \le i \le n\}$, *PCS* checks whether ID_p satisfies m_ω. If it holds, *PCS* accepts and stores them; otherwise, *PCS* refuses to accept them.
- *Proof (PCS, O)*: This is a 2-move interactive protocol between *PCS* and the original client *O*. If O authorizes the remote data integrity checking task to some verifier, it sends (R_o, R_p, R_1) to the authorized verifier. The authorized verifier may be the third auditor or O's proxy. Since *O* has (R_o, R_p, R_1), *O* can performs the interactive protocol *Proof* as the verifier. When the verifier is *O*, the interaction protocol *Proof* is given below.
  1) Challenge (*O*→ PCS): O generates the challenge $chal = (c, k_1, k_2)$. In *chal*, c is the challenged block number which is determined by *O* and k_1, k_2 are randomly picked from $\mathcal{Z}_q^*$. Then, it sends the challenge $chal$ to PCS;
  2) Response (*O*← PCS): Upon receiving O's challenge $chal = (c, k_1, k_2)$, *PCS* performs the following procedures:
     a) Search for the two-tuples (O, k_1) in the table T_{chal}. If $(O, k_1) \in T_{chal}$, *PCS* refuses and informs the requester *O* ; otherwise, it goes on;
     b) For $1 \le i \le c$, *PCS* computes $v_i = \pi_{k_1}(i), a_i = f_{k_2}(i)$ by using k_1, k_2, and

$$T = \prod_{1 \le i \le c} {T_{v_i}}^{a_i}$$

$$\bar{F} = \sum_{1 \le i \le c} a_i F_{v_i}$$

     c) *PCS* sends $\theta = (\bar{F}, T)$ to *O*.
  3) Upon receiving the response $\theta = (\bar{F}, T)$, *O* performs the following procedures:
     a) By using the generated *chal*, *O* computes

$$v_i = \pi_{k_1}(i),\ a_i = f_{k_2}(i)$$

$$h_{v_i} = h(v_i, N_{v_i}),\ u = h(n+1, ID_0)$$

     b) *O* computes

$$\bar{R} = R_1(R_o R_p Y^{H(ID_o, R_o) + H(ID_p, R_p)})^{H(m_\omega, R_1)}$$

     and verifies the following formula:

$$e(T, g) = e(\prod_{i=1}^{c} h_{v_i}^{a_i} u^{\bar{F}}, \bar{R}) \quad (2)$$

     If the formula (2) holds, *O* outputs "success"; otherwise, *O* outputs "failure".
  4) When *O* outputs "failure", it will perform the same challenge many times. If the responses still cannot pass the verification, *O* will inform the *PCS* manager this situation. *PCS* manager will censor *O*'s stored data and retrieve the lost data from the offline backup. If *PCS* manager fails, *O* and *PCS* manager will have to evaluate the loss and discuss the reparation according to the loss severity.

Notes: In *Proof(PCS, O)*, *O* generates the challenge $chal = (c, k_1, k_2)$ where k_1, k_2 are randomly picked from $\mathcal{Z}_q^*$. Based on $chal$, *PCS* gets the challenged block index set as $\{v_1, v_2, \cdots, v_c\}$ where $v_i = \pi_{k_1}(i),\ 1 \le i \le c$. Since $\pi_{k_1}(\cdot)$ is a pseudo-random permutation determined by the random k_1, the challenged block index set $\{v_1, v_2, \cdots, v_c\}$ comes from the random selection of the n stored block-tag pairs.

Correctness: Our proposed concrete ID-PUIC protocol must be workable and correct. Correctness means that, if *KGC*, *PCS*, *O* and *Proxy* are honest and follow the specified procedures, *PCS*'s response θ can pass *O*'s verification. Our protocol's correctness comes from the following deduction:

$$\begin{aligned} & e(T, g) \\ = \ & e(\textstyle\prod_{i=1}^{c} T_{v_i}^{a_i}, g) \\ = \ & e(\textstyle\prod_{i=1}^{c} h(v_i, N_{v_i})^{a_i} u^{a_i F_{v_i}}), g^\sigma) \\ = \ & e(\textstyle\prod_{i=1}^{c} h(v_i, N_{v_i})^{a_i} u^{\sum_{i=1}^{c} a_i F_{v_i}}), g^\sigma) \\ = \ & e(\textstyle\prod_{i=1}^{c} h(v_i, N_{v_i})^{a_i} u^{\bar{F}}, g^\sigma) \\ = \ & e(\textstyle\prod_{i=1}^{c} h_{v_i}^{a_i} u^{\bar{F}}, g^\sigma) \end{aligned}$$

On the other hand, by using the concrete σ, we get

$$\begin{aligned} & g^\sigma \\ = \ & g^{\sigma_1 + \sigma_p H(m_\omega, R_1)} \\ = \ & g^{\sigma_1} g^{\sigma_p H(m_\omega, R_1)} \\ = \ & R_1(R_o Y^{H(ID_o, R_o)})^{H(m_\omega, R_1)} (R_p Y^{H(ID_p, R_p)})^{H(m_\omega, R_1)} \\ = \ & R_1(R_o R_p Y^{H(ID_o, R_o) + H(ID_p, R_p)})^{H(m_\omega, R_1)} \end{aligned}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2520886, IEEE Transactions on Information Forensics and Security

Denote $\bar{R} = R_1(R_o R_p Y^{H(ID_o,R_o)+H(ID_p,R_p)})^{H(m_\omega,R_1)}$. Thus,

$$e(T,g) = e(\prod_{i=1}^{c} h_{v_i}^{a_i} u^{\bar{F}}, \bar{R})$$

Thus, the correctness is proved.

C. Performance analysis

First, we give the computation and communication overhead of our proposed ID-PUIC protocol. At the same time, we implement the prototype of our ID-PUIC protocol and evaluate its time cost. Then, we give the flexibility of remote data integrity checking in the phase *Proof* of our ID-PUIC protocol. At last, we compare our ID-PUIC protocol with the other up-to-date remote data integrity checking protocols.

Computation: Let the uploaded block number be n. Let the challenge be $chal = (c, k_1, k_2)$ where $1 \leq c \leq n, k_1, k_2 \in \mathcal{Z}_q^*$. Based on the different phases, we give the computation overhead. On the group $\mathcal{G}_1$, bilinear pairings, exponentiation, and multiplication contribute most computation cost. Compared with them, the other operations are faster, such as, hash function h, the operations on $\mathcal{Z}_q^*$ and $\mathcal{G}_2$, *etc*. The hash function H can be done once for all. Thus, we only consider bilinear pairings, exponentiation, and multiplication on $\mathcal{G}_1$. For the proxy, the computation overhead mainly comes from the phase $TagGen$. In the phase *TagGen*, the proxy performs $2n$ exponentiation, n multiplication on the group $\mathcal{G}_1$, and n hash function h. In the phase *Proof*, the original client *O* generates the challenge *chal* and *PCS* responds to *chal*. In the interactive proof *Proof*, *PCS* performs c exponentiation and $c-1$ multiplication on the group $\mathcal{G}_1$ and sends the response θ to the checker *O*. In order to check the response θ's validity, *O* performs c+3 exponentiation, 2 pairings, $3+c$ multiplication on the group $\mathcal{G}_1$ and c hash function h.

In order to show our protocol's practical computation overhead, we have simulated the proposed ID-PUIC protocol by using C programming language with GMP Library (GMP-5.0.5) [36], and PBC Library (pbc-0.5.13) [37], [38]. In the simulation, *PCS* works on DELL PowerEdge R420 Server with the following settings:

- CPU: Intel R Xeon R processor E5-2400 and E5-2400 v2 product families
- Physical Memory: 8GB DDR3 1600MHz
- OS: Ubuntu 13.04 Linux 3.8.0-19-generic SMP i686

The proxy and the original client works on PC Laptop with the following settings:

- CPU: CPU I PDC E6700 3.2GHz
- Physical Memory: DDR3 2G 1600MHz
- OS: Windows 7

In the simulation, we choose type A pairing parameters, where the group order is 160 bits, and the order of the base field is 512bit. Type A pairing parameters provide a competitive security level with 1024-bit RSA. In *TagGen*, when one tag is created, proxy's time cost is 0.022472s. When one thousand tags are created, proxy's time cost is 22.567269s. In order to show the time cost, we give Figure 2. Figure 2 depicts proxy's computation cost in *TagGen*. X-label denotes the created tag number n and Y-label denotes the time cost (second) in order to create n tags. Proxy's time cost increases almost linearly with the created tag number n. In *Proof*, the original client interacts with *PCS* and performs the remote data integrity checking. In *Proof*, when the challenged block number is 20, the original client's time cost is 0.567881s. When the challenged block number is 100, the original client's time cost is 2.356659s. In order to show the time cost, we give Figure 3. Figure 3 depicts original client's computation time cost in *Proof*. X-label denotes the challenged block number c and Y-label denotes the time cost (second) in order to check *PCS*'s response θ. At the same time, we consider the computation time cost of *PCS*. In *Proof*, when the challenged block number is 20, the *PCS*'s time cost is 1.611138s. When the challenged block number is 100, the *PCS*'s time cost is 7.993312s. In order to show the time cost, *PCS*'s time cost figure is added to Figure 3. Figure 3 also depicts *PCS*'s computation time cost in *Proof*. X-label denotes the challenged block number c and Y-label denotes *PCS*'s time cost (second) in order to create the response θ. From Figure 3, we know that most computation are performed by *PCS*. The end device only performs a little computation. Our protocol can be used in the environment where the checker's computation resource is limited, for example mobile phone.

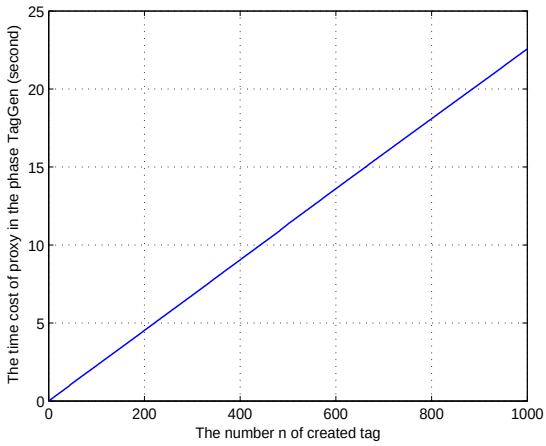


Fig. 2. Proxy's time cost in TagGen (second)

Communication: National Bureau of Standards and ANSI X9 have determined the shortest key length requirements: RSA and DSA is 1024 bits, ECC is 160 bits [35]. According to the above standard, we analyze our ID-PUIC protocol's communication cost. After the data processing, the block-tag pairs are uploaded to *PCS* once and for all. Thus, we only consider the communication cost which is incurred in the remote data integrity checking. In *Proof*, the communication cost comprises the challenge *chal* and response θ. The original client will interact with *PCS* periodically in the phase *Proof*. Suppose there are *n* message blocks are stored in the *PCS*. In order to finish one round interaction, the original client will create the challenge $chal = (c, k_1, k_2)$ and send *chal* to *PCS*. The whole communication cost is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2520886, IEEE Transactions on Information Forensics and Security

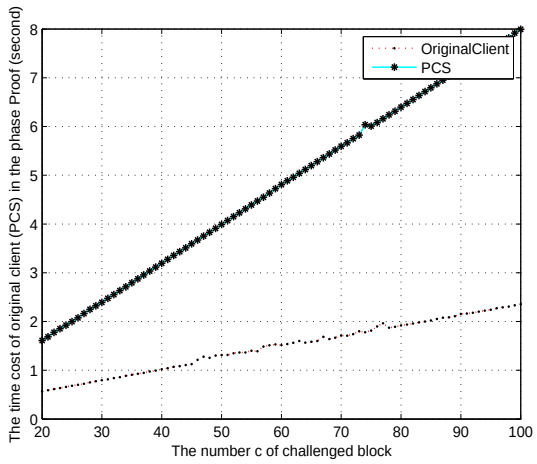


Fig. 3. PCS and OriginalClient's time cost in Proof (second)

$\log_2 n + 2\log_2 q = 320 + \log_2 n$ bits. In order to respond the challenge *chal*, *PCS* creates the response $\theta = (\bar{F}, T)$. θ's bit length is $160 + 1|\mathcal{G}_1| = 160 + 2 * 512 = 1184$ bits. Thus, for one round interaction of *Proof*, the whole communication cost is $320 + \log_2 n + 1184 = 1504 + \log_2 n$ bits.

Private checking, delegated checking and public checking: Our proposed ID-PUIC protocol satisfies the private checking, delegated checking and public checking. In the remote data integrity checking procedure, R_1, R_o, R_p are indispensable. Thus, the procedure can only be performed by the entity who has R_1, R_o, R_p. In general, since R_1, R_o, R_p are kept secret by the original client, our protocol can only be performed by the original client. Thus, it is private checking. On some cases, the original client has no ability to check its remote data integrity, such as, he is taking a vacation or in prison or in battle field, *etc*. Thus, it will delegate the third party to perform the ID-PUIC protocol. It can be the third auditor or the proxy or other entities. The original client sends R_1, R_o, R_p to the delegated third party. The delegated third party has the ability to perform the ID-PUIC protocol. Thus, it has the property of delegated checking. On the other hand, if the original client makes R_1, R_o, R_p public, any entity has the ability to perform the ID-PUIC protocol. Thus, our protocol has also the property of public checking.

Notes: In the checking, the checker must have R_1, R_o, R_p. R_o, R_p are the part of original client's private key and the proxy's private key respectively. Their publicity cannot leak their the other part of private key, *i.e.*, σ_o, σ_p cannot be leaked. The private key extraction phase *Extract* is actually a modified ElGamal signature scheme which is existentially unforgeable. For the identity *ID*, the extracted private key (R, σ) is a signature of *ID*. Since ElGamal signature is existentially unforgeable, the private key part σ will keep secret even if R is made public. On the other hand, R_1 is generated by the original client in order to create the signature on the warrant m_ω. Thus, R_1 is also known to the original client.

Comparison: In order show our ID-PUIC protocol's superiority, we compare our protocol with the recent two protocols, *i.e.*, Wang's protocol [16] and Zhang *et al.*'s protocol [23]. Since most computation cost comes from the pairings, exponentiation and multiplication on the group $\mathcal{G}_1$, the simplified comparison is given in Table II. In the comparison, we denote the time cost of pairings, exponentiation and multiplication as C_p, C_e, C_m, respectively. From the computation comparison, we know that our protocol has almost the same computation cost in the phases *TagGen* and PCS has the same computation cost in the phase *Proof*. For the checker in the phase *Proof*, our protocol has less computation cost than the other two protocols. On the other hand, our protocol can realize the three security properties: proxy data processing and uploading, remote data integrity checking with flexibility and not required certificate management. Flexibility means that our protocol can realize the private checking, delegated checking and public checking according to the original client's authorization.

Hybrid cloud: Our contributions are also suitable for the scenario of hybrid clouds, where the proxy can be treated as the private cloud of the original client. In the scenario, the private cloud gets its private/public key pairs. The private cloud gets the proxy-key and the authorization of the original client through the interaction between the original client and its private cloud. When the original client needs its private cloud perform the data uploading task, it informs its private cloud. Upon receiving the original client's instruction, the private cloud will interact with the public cloud and finish the data uploading task.

IV. Security Analysis

The security of our ID-PUIC protocol mainly consists of the following parts: correctness, proxy-protection and unforgeability. The correctness has been shown in the subsection III-B. In the following paragraph, we study the proxy-protection and unforgeability. Proxy-protection means that the original client cannot pass himself off as the proxy to create the tags. Unforgeability means that when some challenged blocks are modified or deleted, *PCS* cannot send the valid response which can pass the integrity checking.

In order to formally prove our protocol, we give the definition of (t, ϵ)-security below:

Definition 10 ((t, ϵ)-security): A problem is called to satisfy (t, ϵ)-security if no adversary can break it with the probability at least ϵ within the time period t.

Theorem 1 (Proxy-protection): Let $(\mathcal{G}_1, \mathcal{G}_2)$ be a (t', ϵ')-GDH group pair where $\mathcal{G}_1$ and $\mathcal{G}_2$ have the same prime order q. Then, our proposed ID-PUIC protocol is (t, ϵ) proxy-protective in the random oracle model, where t and ϵ satisfies $\epsilon' \geq \frac{1}{\bar{n}}(\frac{q_T}{q_T+1})^{q_T}\frac{1}{q_T+1}\epsilon$ and $t' \leq t + O(q_H) + O(q_h) + O(q_E) + O(q_T) + O(q_p)$. $\bar{n}$ denotes the number of all the identities. In particular, suppose $\mathcal{A}_1$ is an adversary in the random oracle model, and $\mathcal{A}_1$ runs in time t, and makes at most q_H hash *H-Oracle* queries, q_h hash *h-Oracle* query, q_E *Extract-Oracle* query, q_T *TagGen-Oracle* query and q_p *Proxy-key Generation-Oracle* query. Based on the difficulty of CDH problem, our ID-PUIC protocol satisfies the property of proxy-protection.

Proof: Let the selected identity set be $I = \{ID_1, ID_2, \cdots, ID_{\bar{n}}\}$ which includes the proxy identity ID_p.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2520886, IEEE Transactions on Information Forensics and Security

TABLE II
COMPARISON OF COMPUTATION AND SECURITY PROPERTY

Protocols	TagGen	PCS's cost in Proof	Checker's cost in Proof	Proxy Data Processing & Uploading	Integrity Checking Flexibility	Certificate management	Key Escrow
Wang [16]	$1C_p$+$2C_e$+$1C_m$	cC_e+(c-1)C_m	$3C_p$+(c+1)C_e+cC_m	No	No	Required	No
Zhang [23]	$2C_e$+$2C_m$	cC_e+(c-1)C_m	$3C_p$+(c+2)C_e+cC_m	No	No	Required	No
Our protocol	$2C_e$+$1C_m$	cC_e+(c-1)C_m	$2C_p$+(c+1)C_e+cC_m	Yes	Yes	Not Required	Yes

Let the uploaded data be *F* which is divided into *n* blocks, *i.e.,* $F = (F_1, F_2, \cdots, F_n)$, where $F_i \in \mathcal{Z}_q^*$. Suppose that the original client can (t, ϵ)-counterfeit the proxy to generate block-tag pairs. We can construct another algorithm $\mathcal{C}_1$ which can (t', ϵ')-break the CDH problem. Based on the difficulty of CDH problem, the theorem is proved. $\mathcal{C}_1$ will simulate $\mathcal{A}_1$'s attack environment. Denote g as a generator of the group $\mathcal{G}_1$. In the following game, the adversary is denoted as $\mathcal{A}_1$ and the challenger is denoted as $\mathcal{C}_1$. Let (P_1, P_2) be $P_1 = g^a, P_2 = g^b$ where $a, b \in \mathcal{Z}_q^*$ are unknown. Input the tuple (g, P_1, P_2), $\mathcal{C}_1$ will try to calculate g^{ab} by using $\mathcal{A}_1$'s forgery.

Setup: $\mathcal{C}_1$ picks $x \in \mathcal{Z}_q^*$ and calculates $Y = g^x$. Then, it sets the *KGC*'s private key/public key pair is (x, Y). Let m_ω be the warrant which describes the original client's authorization. $\mathcal{C}_1$ calculates $u = h(n+1, ID_o)$ and sends u to $\mathcal{A}_1$. The three tables T_H, T_h, and T_E are initialized to be empty. T_H stores the hash function *H*'s query-response records, T_h stores the hash function *h*'s query-response records and T_E stores the oracle *Extract*'s query-response records.

H-query. $\mathcal{A}_1$ sends $Q_i = (ID_i, R_i)$ or $Q_i = (m_\omega, R_i)$ to $\mathcal{C}_1$. $\mathcal{C}_1$ looks up the table T_H. If there exists $(Q_i, t_i) \in T_H$, $\mathcal{C}_1$ sends t_i to $\mathcal{A}_1$; otherwise, $\mathcal{C}_1$ picks a random $t_i \in \mathcal{Z}_q^*$ and stores (Q_i, t_i) in the table T_H. Then, it outputs t_i to $\mathcal{A}_1$ as the response.

h-query. $\mathcal{A}_1$ sends (i, N_i) to $\mathcal{C}_1$. $\mathcal{C}_1$ responds $\mathcal{A}_1$ below:

1) $\mathcal{C}_1$ looks up the table T_h whose records are the format (i, N_i, z_i, b_i, c_i). If the query (i, N_i) has been stored in the table T_h, $\mathcal{C}_1$ sends $z_i u^{-F_i}$ to $\mathcal{A}_1$, *i.e.,* $h(i, N_i) = z_i u^{-F_i}$.
2) If the query (i, N_i) does not be stored in T_h, $\mathcal{C}_1$ picks a random $b_i \in \mathcal{Z}_q^*$. At the same time, it also picks a random coin $c_i \in \{0, 1\}$ based on the bivariate probability distribution $\Pr[c_i = 0] = \frac{1}{q_T+1}$. If $c_i = 1$, $\mathcal{C}_1$ calculates $z_i = g^{b_i}$; otherwise, $\mathcal{C}_1$ calculates $z_i = P_2 g^{b_i}$. Then, $\mathcal{C}_1$ adds the record (i, z_i, b_i, c_i) to T_h and responds $h(i, N_i) = z_i u^{-F_i}$ to $\mathcal{A}_1$.

Extract: $\mathcal{A}_1$ sends ID_i to $\mathcal{C}_1$ where $ID_i \neq ID_p$. Since $\mathcal{C}_1$ has KGC's private key x, it can calculate ID_i's private key (R_i, σ_i) by using the phase *Extract* of our *ID-PUIC* protocol. Especially, $\mathcal{A}_1$ can query the original client's private key and get (R_o, σ_o) from $\mathcal{C}_1$. In T_E, the corresponding record can be denoted as (Q_o, H_o, σ_o) where $Q_o = (ID_o, R_o)$, $H_o = H(ID_o, R_o)$. On the other hand, although $\mathcal{A}_1$ cannot query the proxy ID_p's private key, but $\mathcal{C}_1$ can generate part of ID_p's private key below: Picks random $H_p \in \mathcal{Z}_q^*$ and calculates $R_p = P_1 Y^{-H_p}$. Then, $\mathcal{C}_1$ sets $H_p = H(ID_p, R_p)$. It is obvious that $\sigma_p = a$ is unknown.

Proxy-key Generation query. $\mathcal{A}_1$ sends (ID'_o, ID'_p) to $\mathcal{C}_1$. If $ID'_p \neq ID_p$, ID'_p and ID'_o's private keys can be extracted in the phase *Extract*. Thus, proxy-key can be generated by using the normal procedure. Otherwise, aborts.

Notes: Since $\mathcal{A}_1$ has gotten the original client's private key, it can generates the warrant m_ω's signature $(m_\omega, R_1, \sigma_1)$ and sends it to $\mathcal{C}_1$. In the practical scenario, the proxy can also get the warrant's signature. Our assumption is appropriate.

TagGen query. $\mathcal{A}_1$ sends (i, N_i, F_i) to $\mathcal{C}_1$. $\mathcal{C}_1$ performs the following procedures: If (i, N_i, z_i, b_i, c_i) has not been stored in the table T_h, $\mathcal{A}_1$ sends the query (i, N_i) to *h-Oracle*. $\mathcal{C}_1$ responds $\mathcal{A}_1$ and adds the record (i, N_i, z_i, b_i, c_i) to T_h. Otherwise, $\mathcal{C}_1$ performs the following procedure. If $c_i = 0$, $\mathcal{C}_1$ reports failure and terminates; otherwise, $\mathcal{C}_1$ calculates

$$T_i = g^{\sigma_1 b_i} (P_1)^{b_i H(m_\omega, R_1)}$$

and sends T_i to $\mathcal{A}_1$.

Notes: $\mathcal{C}_1$'s response to the tag query is valid from the derivation :

$$\begin{aligned} T_i &= g^{\sigma_1 b_i} (P_1)^{b_i H(m_\omega, R_1)} \\ &= (g^{\sigma_1} g^{aH(m_\omega, R_1)})^{b_i} \\ &= (g^{\sigma_1} g^{\sigma_p H(m_\omega, R_1)})^{b_i} \\ &= (g^{\sigma})^{b_i} \\ &= (g^{b_i})^{\sigma} \\ &= (h(i, N_i) u^{F_i})^{\sigma} \end{aligned}$$

Output: At last, $\mathcal{A}_1$ outputs a forged block-tag pair (F_f, T_f) where the block F_f's index-name pair is (f, N_f). $\mathcal{C}_1$ looks up the table T_h and gets the record (f, N_f, z_f, b_f, c_f) (If the record does not exist, $\mathcal{C}_1$ calculates and adds it to T_h based on the phase *h-query*). When $c_f = 1$, the game aborts and fails; otherwise,

$$\begin{aligned} T_f &= (h(f, N_f) u^{F_f})^{\sigma} \\ &= (P_2 g^{b_f})^{\sigma} \\ &= (P_2 g^{b_f})^{\sigma_1} (P_2 g^{b_f})^{aH(m_\omega, R_1)} \end{aligned}$$

We can get

$$g^{ab} = P_2^a = [T_f (P_2 g^{b_f})^{-\sigma_1} P_1^{-b_f H(m_\omega, R_1)}]^{\frac{1}{H(m_\omega, R_1)}}$$

Thus, CDH problem is solved.

Probability Analysis. Now, we evaluate $\mathcal{C}_1$'s success probability. In order to break CDH problem, the following four events must hold simultaneously:

1) $\mathcal{E}_1$: $\mathcal{C}_1$ does not abort in $\mathcal{A}_1$'s *Proxy-key Generation* query.
2) $\mathcal{E}_2$: $\mathcal{C}_1$ does not abort in $\mathcal{A}_1$'s *TagGen* query.
3) $\mathcal{E}_3$: $\mathcal{A}_1$ generates a valid block-tag forgery (F_f, T_f) which satisfies $c_f = 0$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2520886, IEEE Transactions on Information Forensics and Security

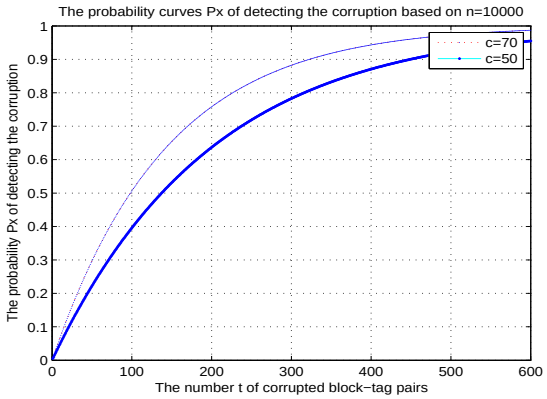


Fig. 4. The probability curve P_X of detecting the corruption

$\mathcal{C}_1$ succeeds if the above three events hold. We calculate the following probability:

$$\begin{aligned} & \Pr[\mathcal{E}_1 \bigwedge \mathcal{E}_2 \bigwedge \mathcal{E}_3] \\ = \ & \Pr[\mathcal{E}_1] \Pr[\mathcal{E}_2|\mathcal{E}_1] \Pr[\mathcal{E}_3|\mathcal{E}_1 \bigwedge \mathcal{E}_2] \\ = \ & \frac{1}{\bar{n}}(\frac{q_T}{q_T+1})^{q_T} \frac{1}{q_T+1}\epsilon \end{aligned}$$

In $\mathcal{C}_1$'s simulation environment, we assume that $(\mathcal{G}_1, \mathcal{G}_2)$ is (t', ϵ')-secure GDH group. We can get

$$\epsilon' \geq \Pr[\mathcal{E}_1 \bigwedge \mathcal{E}_2 \bigwedge \mathcal{E}_3] = \frac{1}{\bar{n}}(\frac{q_T}{q_T+1})^{q_T} \frac{1}{q_T+1}\epsilon$$

$\mathcal{C}_1$'s running time is the same as $\mathcal{A}_1$'s running time plus the time which is spent to respond q_H *H-query*, q_h *h-query*, q_E *Extract-query*, q_T *TagGen-query* and q_p *Proxy-key Generation-query*. Thus, $\mathcal{C}_1$'s whole running time is at most $t' \leq t + O(q_H) + O(q_h) + O(q_E) + O(q_T) + O(q_p)$. Thus, if $\mathcal{A}_1$ can (t, ϵ)-forge block-tag pair, $\mathcal{C}_1$ can (t', ϵ')-break the GDH group. It contradicts the assumption that $(\mathcal{G}_1, \mathcal{G}_2)$ is a (t', ϵ')-GDH group pair. The proof is completed. ■

In our ID-PUIC protocol, the phase *TagGen* is the same as the phase *Pub.St* of PoRs construction for public verifiability [20]. Based on the protocol [20]'s unforgeability, our ID-PUIC protocol satisfies the unforgeability.

Theorem 2 (Unforgeability): The proposed ID-PUIC protocol is existentially unforgeable in the random oracle model if CDH problem on $\mathcal{G}_1$ is hard.

Since the proof process is almost the same as Shacham-Waters's protocol [20], we only give the differences. In Shacham-Waters's protocol, u is randomly picked from $\mathcal{G}_1$. In our ID-PUIC protocol, u is calculated by using the hash function *h*. In the random oracle model, *h*'s output value is indistinguishable from a random value nn the group $\mathcal{G}_1$. In the phase *TagGen*, the proxy-key σ is used in ID-PUIC protocol while the data owner's secret key a is used in Shacham-Waters's protocol [20]. For *PCS*, σ and *a* has the same function to generate the block tags. When *PCS* is dishonest, since Shacham-Waters's protocol is existentially unforgeable in random oracle model, our proposed ID-PUIC protocol is also existentially unforgeable in the random oracle model. The detailed proof process is omitted since it is very similar to Shacham-Waters's protocol.

Theorem 3: Let the uploaded block-tag pairs number be *n*. When $\bar{d}$ block-tag pairs are broken and c block-tag pairs are challenged, the broken block-tag pairs can be found out with probability at least $1 - (\frac{n-\bar{d}}{n})^c$ and at most $1 - (\frac{n-c+1-\bar{d}}{n-c+1})^c$. Thus, our ID-PUIC protocol is $(\frac{\bar{d}}{n}, 1 - (\frac{n-\bar{d}}{n})^c)$-secure, *i.e.*,

$$1 - (\frac{n-\bar{d}}{n})^c \leq P_X \leq 1 - (\frac{n-c+1-\bar{d}}{n-c+1})^c$$

where P_X denotes the probability of detecting the modification.

Proof: Let the challenged block-tag pair set be $\mathcal{S}$ and the modified block-tag pair set be $\mathcal{M}$. Let the random variable X be $X = |\mathcal{S} \bigcap \mathcal{M}|$. According to the definition of P_X, we can get

$$\begin{aligned} P_X &= P\{X \geq 1\} \\ &= 1 - P\{X = 0\} \\ &= 1 - \frac{n-\bar{d}}{n}\frac{n-1-\bar{d}}{n-1} \cdot \cdots \cdot \frac{n-c+1-\bar{d}}{n-c+1} \end{aligned}$$

Thus, we can get

$$1 - (\frac{n-\bar{d}}{n})^c \leq P_X \leq 1 - (\frac{n-c+1-\bar{d}}{n-c+1})^c$$

This completes the proof of the theorem. ■

We give the probability curve of P_X in Figure 4. Combined with the performance analysis, when the stored block-tag pairs number be 10000 and $\bar{d} = 500, c = 70$, the probability is 97.28%, the original client's time cost is 1.920119s and *PCS*'s time cost is 6.397231s. When $n = 10000, \bar{d} = 500, c = 50$, the probability is 92.36%, the original client's time cost is 1.715752s and *PCS*'s time cost is 5.516639s. From our experiments, our ID-PUIC protocol is efficient and practical.

V. Conclusion

Motivated by the application needs, this paper proposes the novel security concept of ID-PUIC in public cloud. The paper formalizes ID-PUIC's system model and security model. Then, the first concrete ID-PUIC protocol is designed by using the bilinear pairings technique. The concrete ID-PUIC protocol is provably secure and efficient by using the formal security proof and efficiency analysis. On the other hand, the proposed ID-PUIC protocol can also realize private remote data integrity checking, delegated remote data integrity checking and public remote data integrity checking based on the original client's authorization.

ACKNOWLEDGMENTS

The author sincerely thanks the Editor for allocating qualified and valuable referees. The author sincerely thanks the anonymous referees for their very valuable comments. The work of H. Wang was supported by the National Natural Science Foundation of China (No. 61272522), the Natural Science Foundation of Liaoning Province (No. 2014020147) and the Program for Liaoning Excellent Talents in University (No. LR2014021). The work of D. He was supported by the National Natural Science Foundation of China (Nos. 61572379, 61501333) and the Natural Science Foundation of

Hubei Province of China (No. 2015CFB257). The work of S. Tang was supported by 973 Program (No. 2014CB360501), and Guangdong Provincial Natural Science Foundation (No. 2014A030308006).

REFERENCES

[1] Z. Fu, X. Sun, Q. Liu, L. Zhou, J. Shu, "Achieving efficient cloud search services: multi-keyword ranked search over encrypted cloud data supporting parallel computing," *IEICE Transactions on Communications*, vol. E98-B, no. 1, pp.190-200, 2015.

[2] Y. Ren, J. Shen, J. Wang, J. Han, S. Lee, "Mutual verifiable provable data auditing in public cloud storage," *Journal of Internet Technology*, vol. 16, no. 2, pp. 317-323, 2015.

[3] M. Mambo, K. Usuda, E. Okamoto, "Proxy signature for delegating signing operation", *CCS 1996*, pp. 48C57, 1996.

[4] E. Yoon, Y. Choi, C. Kim, "New ID-based proxy signature scheme with message recovery", *Grid and Pervasive Computing*, LNCS 7861, pp. 945-951, 2013.

[5] B. Chen, H. Yeh, "Secure proxy signature schemes from the weil pairing", *Journal of Supercomputing*, vol. 65, no. 2, pp. 496-506, 2013.

[6] X. Liu, J. Ma, J. Xiong, T. Zhang, Q. Li, "Personal health records integrity verification using attribute based proxy signature in cloud computing", *Internet and Distributed Computing Systems*, LNCS 8223, pp. 238-251, 2013.

[7] H. Guo, Z. Zhang, J. Zhang, "Proxy re-encryption with unforgeable re-encryption keys", *Cryptology and Network Security*, LNCS 8813, pp. 20-33, 2014.

[8] E. Kirshanova, "Proxy re-encryption from lattices", *PKC 2014*, LNCS 8383, pp. 77-94, 2014.

[9] P. Xu, H. Chen, D. Zou, H. Jin, "Fine-grained and heterogeneous proxy re-encryption for secure cloud storage", *Chinese Science Bulletin*, vol.59, no.32, pp. 4201-4209, 2014.

[10] S. Ohata, Y. Kawai, T. Matsuda, G. Hanaoka, K. Matsuura, "Re-encryption verifiability: how to detect malicious activities of a proxy in proxy re-encryption", *CT-RSA 2015*, LNCS 9048, pp. 410-428, 2015.

[11] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, D. Song, "Provable data possession at untrusted stores", *CCS'07*, pp. 598-609, 2007.

[12] G. Ateniese, R. DiPietro, L. V. Mancini, G. Tsudik, "Scalable and efficient provable data possession", *SecureComm 2008*, 2008.

[13] C. C. Erway, A. Küpçü, C. Papamanthou, R. Tamassia, "Dynamic provable data possession", *CCS'09*, pp. 213-222, 2009.

[14] E. Esiner, A. Küpçü, Ö Özkasap, "Analysis and optimization on FlexDPDP: a practical solution for dynamic provable data possession", *Intelligent Cloud Computing*, LNCS 8993, pp. 65-83, 2014.

[15] E. Zhou, Z. Li, "An improved remote data possession checking protocol in cloud storage", *Algorithms and Architectures for Parallel Processing*, LNCS 8631, pp. 611-617, 2014.

[16] H. Wang, "Proxy provable data possession in public clouds," *IEEE Transactions on Services Computing*, vol. 6, no. 4, pp. 551-559, 2013.

[17] H. Wang, "Identity-based distributed provable data possession in multicloud storage", *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 328-340, 2015.

[18] H. Wang, Q. Wu, B. Qin, J. Domingo-Ferrer, "FRR: Fair remote retrieval of outsourced private medical records in electronic health networks", *Journal of Biomedical Informatics*, vol. 50, pp. 226-233, 2014.

[19] H. Wang, "Anonymous multi-receiver remote data retrieval for pay-tv in public clouds", *IET Information Security*, vol. 9, no. 2, pp. 108-118, 2015.

[20] H. Shacham, B. Waters, "Compact proofs of retrievability", *ASIACRYPT 2008*, LNCS 5350, pp. 90-107, 2008.

[21] Q. Zheng, S. Xu, "Fair and dynamic proofs of retrievability", *CODASPY'11*, pp. 237-248, 2011.

[22] D. Cash, A. Küpçü, D. Wichs, "Dynamic proofs of retrievability via oblivious ram", *EUROCRYPT 2013*, LNCS 7881, pp. 279-295, 2013.

[23] J. Zhang, W. Tang, J. Mao, "Efficient public verification proof of retrievability scheme in cloud", *Cluster Computing*, vol. 17, no. 4, pp. 1401-1411, 2014.

[24] J. Shen, H. Tan, J. Wang, J. Wang, S. Lee, "A novel routing protocol providing good transmission reliability in underwater sensor networks", *Journal of Internet Technology*, vol. 16, no. 1, pp. 171-178, 2015.

[25] T. Ma, J. Zhou, M. Tang, Y. Tian, Al-dhelaan A., Al-rodhaan M., L. Sungyoung, "Social network and tag sources based augmenting collaborative recommender system", *IEICE Transactions on Information and Systems*, vol.E98-D, no.4, pp. 902-910, 2015.

[26] K. Huang, J. Liu, M. Xian, H. Wang, S. Fu, "Enabling dynamic proof of retrievability in regenerating-coding-based cloud storage", *ICC 2014*, pp.712-717, 2014.

[27] C. Wang, Q. Wang, K. Ren, W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing", *INFOCOM 2010*, pp. 1-9, 2010.

[28] Q. Wang, C. Wang, K. Ren, W. Lou, J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing", *IEEE Transactions on Parallel And Distributed Systems* , vol. 22, no. 5, pp. 847-859, 2011.

[29] C. Wang, Q. Wang, K. Ren, N. Cao, W. Lou, "Toward secure and dependable storage services in cloud computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 220-232, 2012.

[30] Y. Zhu, G. Ahn, H. Hu, S. Yau, H. An, S. Chen, "Dynamic Audit Services for Outsourced Storages in Clouds," *IEEE Transactions on Services Computing*, vol. 6, no. 2, pp. 227-238, 2013.

[31] O. Goldreich, "Foundations of cryptography: basic tools", *Publishing House of Electronics Industry*, Beijing, pp. 194-195, 2003.

[32] D. Boneh, B. Lynn, H. Shacham, "Short signatures from the weil pairing", *ASIACRYPT 2001*, LNCS 2248, pp. 514-532, 2001.

[33] D. Boneh, M. Franklin, "Identity-based encryption from the weil pairing", *CRYPTO 2001*, LNCS 2139, pp. 213-229, 2001.

[34] A. Miyaji, M. Nakabayashi, S. Takano, "New explicit conditions of elliptic curve traces for fr-reduction", *IEICE Transactions Fundamentals*, vol. 5, pp. 1234-1243, 2001.

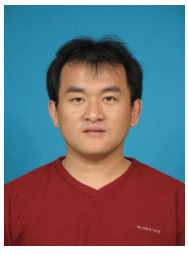
[35] C. Research, "SEC 2: Recommended elliptic curve domain parameters", http://www.secg.org/collateral/sec_final.pdf

[36] The GNU multiple precision arithmetic library (GMP). Available: http://gmplib.org/.

[37] The pairing-based cryptography library (PBC). Available: http://crypto.stanford.edu/pbc/howto.html.

[38] Lynn B., "On the implementation of pairing-based cryptosystems", *Ph.D. dissertation*, http://crypto.stanford.edu/pbc/thesis.pdf, Stanford University, 2008.

Huaqun Wang received the BS degree in mathematics education from the Shandong Normal University and the MS degree in applied mathematics from the East China Normal University, both in China, in 1997 and 2000, respectively. He received the Ph.D. degree in Cryptography from Nanjing University of Posts and Telecommunications in 2006. He is currently a professor of Nanjing University of Posts and Telecommunications, China. His research interests include applied cryptography, network security, and cloud computing security. He has published more than 40 papers. He has served in the program committee of several international conferences and the editor board of international journals.



Debiao He received his Ph.D. degree inapplied mathematics from School of Mathematics and Statistics, Wuhan University in 2009. He is currently an Associate Professor of the State Key Lab of Software Engineering, Computer School, Wuhan University, Wuhan, China. His main research interests include cryptography and information security, in particular,cryptographic protocols.

Shaohua Tang received the B.Sc. and M.Sc. Degrees in applied mathematics, and the Ph.D. Degree in communication and information system all from the South China University of Technology, in 1991, 1994, and 1998, respectively. He has been a full professor with the School of Computer Science and Engineering, South China University of Technology since 2004. His current research interests include information security, networking, and information processing.