Building Secure and Anonymous Communication Channel: Formal Model and its Prototype Implementation

Keita Emura
National Institute of Information and Communications Technology, Japan
k-emura@nict.go.jp

Akira Kanaoka
National Institute of Information and Communications Technology, and Toho University, Japan
akira.kanaoka@is.sci.toho-u.ac.jp

Satoshi Ohta
National Institute of Information and Communications Technology, Japan
sota@nict.go.jp

Takeshi Takahashi
National Institute of Information and Communications Technology, Japan
takeshi_takahashi@ieee.org

ABSTRACT

Various techniques need to be combined to realize anonymously authenticated communication. Cryptographic tools enable anonymous user authentication while anonymous communication protocols hide users' IP addresses from service providers. One simple approach for realizing anonymously authenticated communication is their simple combination, but this gives rise to another issue; how to build a secure channel. The current public key infrastructure cannot be used since the user's public key identifies the user. To cope with this issue, we propose a protocol that uses identity-based encryption for packet encryption without sacrificing anonymity, and group signature for anonymous user authentication. Communications in the protocol take place through proxy entities that conceal users' IP addresses from service providers. The underlying group signature is customized to meet our objective and improve its efficiency. We also introduce a proof-of-concept implementation to demonstrate the protocol's feasibility. We compare its performance to SSL communication and demonstrate its practicality, and conclude that the protocol realizes secure, anonymous, and authenticated communication between users and service providers with practical performance.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*; H.1.m [**Information System**]: Models and Principles—*Miscellaneous*; E.3 [**Data**]: Data Encryption—*Public key cryptosystems*

General Terms

Security, Design, Theory

This is a preprint version of our paper presented in SAC'14, March 24-28, 2014, Gyeongju, Korea.

Keywords

Anonymous Communication, Anonymous Authentication, Secure Channel, Identity-Based Encryption, Group Signature

1. INTRODUCTION

Anonymity[1] is an important aspect of privacy, and systems that provide services to anonymous users are currently a topic of keen interest. Such systems can provide services to users without revealing their identity. To date, a great deal of studies have been reported on [2], and many use cryptography as the important building block for constructing the systems, but these need further improvement before they can be used for actual services. To realize secure, anonymous, and authenticated communication, these building blocks need to collaborate with each other.

1.1 Research Background

Several cryptographic primitives providing anonymity have been proposed. Among them is group signature [10], which enables a signer to anonymously prove signatures' validity. A group manager (GM) that has a pair of a group public key, gpk, and master secret key, msk, issues a secret signing key, sk_i, to a user U_i, which computes a group signature, σ (on certain messages), using sk_i. No user-dependent value is required in the verification phase; a verifier verifies σ using only the corresponding gpk. These approaches alone, however, cannot guarantee anonymity when applied to online communication. For instance, let a signer compute a group signature and *send* it to a verifier. The verifier can anonymously verify the signature's validity. However, there is a question of how to anonymously send the group signature to the verifier. Usually, a source IP address is included in a packet, and that reveals the identity of the sender, thus user anonymity is already infringed upon. The situation remains the same regardless of the primitives we implement so long as direct communication between a sender (signer, prover, etc) and a receiver (verifier, etc) is required.

The user's IP address is naturally visible in the IP packets sent from the user, and it cannot simply be erased or forged

[1]This paper considers sender/prover anonymity and does not consider recipient anonymity.

to enable bi-directional communication. One approach for this is using intermediate agents that send packets on behalf of the actual user terminal, and several such protocols have already been proposed [2], including Tor [5]. Nevertheless, another issue arises in the question of how we can assure user's legitimacy. We need to discern legitimate and illegitimate users to restrict unauthorized access to the channel. One might think that only end-to-end authentication is needed, but it is hard to authenticate users without identifying them. For instance, a server needs to send a response code to a user in basic authentication and the user needs to return a user ID and password. That is, the server needs to identify the user. Moreover, it seems to be hard to send a certain message back from the server to a user since the corresponding source IP address is generally required. Authentication by an intermediate agent (as in Tor [5]) might be a solution to these problems. The agent can authenticate a user and can hide the user's source IP address from the server. That notwithstanding, we still need to know how the server can directly authenticate end users.

A simple approach to the anonymous authentication problems is just combining both cryptographic primitives and anonymous communication protocols as follows. Let a user compute an anonymously-authenticated token (e.g., group signature), and send it to a server via an anonymous channel (e.g., using Tor). Then, the server can directly authenticate the user without compromising anonymity. However, another problem arises is how we can establish a secure channel (i.e., flowed data is encrypted). If the server uses a user's public key (certified by a trusted Certificate Authority (CA) in a public key infrastructure (PKI)), then server identifies the user, since a certificate contains information on the key holder. The same problem arises even if symmetric key encryption is used. Assume that the server tries to exchange a secret key with an end user. Since the server does not know who the actual end user is, the server does not know the user's public key for running a key exchange protocol.

1.2 Our Contribution

We propose a protocol that realizes secure, anonymous, and authenticated communication. The proposed protocol uses identity-based encryption (IBE) to encrypt content without identifying key holders[2]. It can set arbitrary values on public keys, thus it can enable a user to select a temporary ID for each session, which the server can use as a public key. It also uses group signature for anonymous user authentication. Communications in the protocol take place through proxy entities that conceal users' IP addresses from service providers (SPs).

This paper gives the framework of the proposed protocol, gives a formal model and security definitions of the proposed protocol, points out the needlessness of the group signature's open capability for our usage, and then proposes an open-free variant of the Furukawa-Imai group signature scheme [13]. The modification can reduce its signature size by 50% compared to the original scheme. Note that if someone needs to identify an illegitimate user, we can add such a mechanism without relying on cryptographic techniques; e.g., an IP address managed by the proxy.

We demonstrate the feasibility and practicality of the proposed protocol by introducing our proof-of-concept implementation. The implementation uses the modified group signature scheme mentioned above. It also uses the Boneh-Franklin IBE scheme [9] for its underlying IBE scheme.

Note that, our protocol in this paper focuses on encrypted communication from the SP to users. It can easily be extended to interactive secure communication since SP is not anonymous to users and each user thus can simply use SP's public key for building a secure channel.

1.3 Related Work

There exist similar attempts to our approach. Sudarsono et al. [21] has considered an anonymous IEEE802.1X authentication system by using a group signature scheme. They use group signatures as the client's digital certificate. The means of sending such certificates over IP networks was, however, outside its scope. Lee et al. [16] proposed an anonymous subscription service, called Anon-pass. Their construction methodology is similar to group signatures, wherein a user proves the possession of signatures using zero-knowledge proofs. Though Anon-pass does not consider end-to-end secure (encrypted) communication, our protocol does.

Gilad and Herzberg [14] also considered how to distribute public keys using an anonymous service. They consider two peers, a querier and a responder. The querier specifies a random ephemeral public key that is not certified by the CA, and sends a query containing this public key to a responder via an anonymous service, like Tor. The responder replies with a response message encrypted by the (anonymous) querier's ephemeral public key. However, a responder cannot check whether a public key is a valid key or a random value since this scheme gives no certification of the public key, and moreover the responder cannot detect even if the pubic key is replaced by an attacker. Moreover, no anonymous user authentication is considered in the Gilad-Herzberg system. In our protocol, the SP can be convinced that a public key (i.e., a temporary ID) will work, since arbitrary values can be public keys in IBE systems. Moreover, since a temporary ID is signed by group signature, we can prevent the key replacement attack and can achieve anonymous user authentication, simultaneously.

Proxy re-encryption (PRE) (e.g., [18]) is another candidate. In our context, first users compute re-encryption keys using their secret key and the SP public key, and the SP only computes ciphertext using its own public key. Then, the proxy can re-encrypt ciphertexts. However, the proxy needs to manage all re-encryption keys, and therefore it is difficult to assume that no private information is infringed on even if the proxy is corrupted after the communication. Moreover, there is a possibility that other user may decrypt unexpected ciphertexts, since the proxy manages many re-encryption keys (from the SP to each user). It is undesirable to generate re-encryption keys that can be used in an unexpected manner, even if the proxy is modeled as an honest-but-curious entity and always follows the protocol. In our protocol no unexpected user (including the proxy) can decrypt ciphertexts, since a unique temporary ID is assigned for each user and each session. Note that the key escrow

[2]The conventional public key encryption (PKE) with certain non-interactive zero-knowledge (NIZK) proofs may also be applicable, where a user chooses a temporary public key for each session, and makes a NIZK proof of the corresponding secret key. We do not consider this construction anymore since the NIZK proof must be constructed from scratch by considering algebraic structures of the underlying PKE scheme, and this may lead to some difficulty of its implementation.

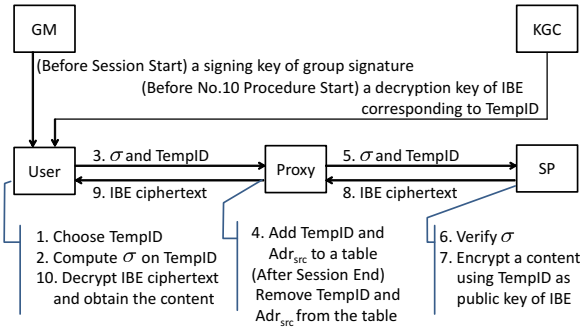

Figure 1: Framework of the Proposed Protocol

problem happens as an outcome of IBE, where key generation center (KGC) can decrypt all ciphertexts. However, KGC is modeled as a trusted third party, whereas it is difficult to fully trust all proxies involved in the systems.

2. FRAMEWORK

Figure 1 depicts the framework of the proposed protocol, which has five roles; User, Proxy, Service Provider, GM, and KGC. A **User** wishes to communicate with an SP without revealing its identity. The **Proxy** assists communication between a User and SP by relaying packets without revealing the User's IP address. We assume that it is honest-but-curious. The **SP** provides services to Users, but wishes to authenticate them. It does not care about their identities but needs to confirm whether the user accessing it is legitimate. The **GM** manages a group key and issuer key, and issues a signing key for a user that is used for generating an anonymously-authenticated token. We assume that the GM suitably authenticates a user before issuing the signing key. The **KGC** generates a decryption key for a user. We assume that the KGC suitably authenticates a user before issuing the decryption key.

These roles need to collaborate with each other to realize the proposed secure anonymous authentication. Their interaction sequence is as follows. (1) A user (whose IP address is $\mathtt{Adr}_{\mathtt{src}}$) chooses a temporary ID $\mathtt{TempID}$, (2) computes a group signature σ on $\mathtt{TempID}$, and (3) sends $(\sigma, \mathtt{TempID})$ to the proxy. (4) The proxy associates $\mathtt{Adr}_{\mathtt{src}}$ with this temporary ID, and (5) forwards $(\sigma, \mathtt{TempID})$ to the SP. (6) The SP can directly authenticate the users by verifying the group signature without compromising anonymous communications. (7) If the user is successfully verified, the SP encrypts content using TempID as the public key of IBE; otherwise it returns $\perp$. Here, we apply an IBE's property to establish a secure channel between the SP and an anonymous user, where arbitrary values can be a public key, and a ciphertext can be independently computed with the generation of the corresponding decryption key[3]. (8) The SP sends this IBE ciphertext to the proxy, which again (9) forwards it to the corresponding user. (10) Finally, the user decrypts the IBE ciphertext using the corresponding decryption key issued by the KGC. After relaying the (mutual) communication, the proxy immediately deletes the corresponding pair of $(\mathtt{TempID}, \mathtt{Adr}_{\mathtt{src}})$. Therefore, no private information is infringed on even if the proxy is corrupted after the communication. Note that Figure 1 explains one-pass communication. The proxy can reuse the information of the pair $(\mathtt{TempID}, \mathtt{Adr}_{\mathtt{src}})$ of a session so long as the session is alive, but it removes the information from its registry once the session is closed. In either case, the proxy immediately deletes the corresponding pair $(\mathtt{TempID}, \mathtt{Adr}_{\mathtt{src}})$ after the session. Moreover, we can easily extend one-proxy setting to multi-proxy setting, since all the proxy has to do is (1) manage $(\mathtt{TempID}, \mathtt{Adr}_{\mathtt{src}})$, and (2) forwarding $(\sigma, \mathtt{TempID})$ to the next. The above framework is embodied as a concrete protocol in the following section.

[3]This property is used in timed-release encryption [11] context, where an encryptor can control when ciphertexts will be decrypted.

3. AUTHENTICATION PROTOCOL

This section proposes a secure anonymous authentication protocol. It first defines the syntax of the protocol, and then gives its construction. We consider a scenario in which an SP is modeled as a server, which provides a service only for a legitimate user. That is, we can assume that the GM has authenticated a user before issuing a signing key, and the SP can judge that the user who can generate a valid group signature is a legitimate user.

3.1 Syntax and Security Definitions

Let $\mathcal{ID}$ and $\mathcal{M}$ be an identity space and message space, respectively, and $\mathtt{Adr}_{\mathtt{src}}$, $\mathtt{Adr}_{\mathtt{proxy}}$, and $\mathtt{Adr}_{\mathtt{dst}}$ stand for IP address of User, Proxy, and SP, respectively. For a set X and an element $x \in X$, $x \xleftarrow{\$} X$ means that x is randomly chosen from X.

Definition 3.1 (Syntax of The Protocol).

GM.Setup : *This probabilistic algorithm takes as input the security parameter λ, and outputs a group public key gpk and an issuer key ik.*

KGC.Setup : *This probabilistic algorithm takes as input the security parameter λ, and outputs a public key $params$ and a master secret key msk.*

Join : *This probabilistic algorithm takes as input gpk and ik, and outputs a signing key sk.*

UserKeyGen : *This (possibly) probabilistic algorithm takes as input $params$, msk, and an (possibly temporary) identity $\mathtt{TempID} \in \mathcal{ID}$, and outputs a decryption key $dk_{\mathtt{TempID}}$.*

SendRequest : *This probabilistic algorithm takes as input gpk, sk, $\mathtt{TempID}$, a source IP address $\mathtt{Adr}_{\mathtt{src}}$, a destination IP address $\mathtt{Adr}_{\mathtt{dst}}$, and a proxy IP address $\mathtt{Adr}_{\mathtt{proxy}}$, and send a token σ, $\mathtt{TempID}$ and $\mathtt{Adr}_{\mathtt{dst}}$ to the proxy whose IP address is $\mathtt{Adr}_{\mathtt{proxy}}$.*

RelayRequest : *This deterministic algorithm takes as input $\mathtt{Adr}_{\mathtt{src}}$, $\mathtt{Adr}_{\mathtt{dst}}$, an ID/IP table $\mathtt{Tbl}$, σ, and $\mathtt{TempID}$, and relays a pair $(\sigma, \mathtt{TempID})$ and $\mathtt{Adr}_{\mathtt{proxy}}$ to the destination SP whose IP address is $\mathtt{Adr}_{\mathtt{dst}}$. Moreover, append $(\mathtt{TempID}, \mathtt{Adr}_{\mathtt{src}})$ to $\mathtt{Tbl}$.*

ValidityCheck : *This deterministic algorithm takes as input gpk, σ, and $\mathtt{TempID}$, and outputs 1 if σ is valid, and 0, otherwise.*

SendContent : *This probabilistic algorithm takes as input gpk, σ, $\mathtt{TempID}$, a content to be sent $M \in \mathcal{M}$, and $\mathtt{Adr}_{\mathtt{proxy}}$, computes a ciphertext C if the token σ is valid, and sends C to the proxy whose IP address is $\mathtt{Adr}_{\mathtt{proxy}}$. Otherwise, if σ is invalid, then return $\perp$.*

RelayContent : *This deterministic algorithm takes as input C and* $\mathtt{Tbl}$*, and relays C to a user whose IP address is* $\mathtt{Adr}_{\mathtt{src}}$ *contained in* $\mathtt{Tbl}$*. Moreover, remove* $(\mathtt{TempID}, \mathtt{Adr}_{\mathtt{src}})$ *from* $\mathtt{Tbl}$*. We assume that the proxy can decide the corresponding source IP address to be relayed by* C[4]*.*

GetContent : *This deterministic algorithm takes as input C and $dk_{\mathtt{TempID}}$, and return M.*

Next, we give formal security definitions as follows. First, we define correctness that guarantees σ is valid and a user always can obtain the corresponding content if all values are honestly generated according to the algorithms.

Definition 3.2 (Correctness). *For all* $(gpk, ik) \leftarrow \mathsf{GM.Setup}(1^\lambda)$, $(params, msk) \leftarrow \mathsf{KGC.Setup}(1^\lambda)$, $sk \leftarrow \mathsf{Join}(gpk, ik)$, $\mathtt{TempID} \in \mathcal{ID}$, $M \in \mathcal{M}$, *and* $(\mathtt{Adr}_{\mathtt{src}}, \mathtt{Adr}_{\mathtt{dst}}, \mathtt{Adr}_{\mathtt{proxy}})$,

$$\Pr[M \leftarrow \mathsf{GetContent}\Big(\mathsf{RelayContent}\big(C, \mathtt{Tbl}\big), \mathsf{UserKeyGen}(param, msk, \mathtt{TempID})\Big)] = 1, \text{ and}$$

$$\Pr[1 \leftarrow \mathsf{ValidityCheck}(gpk, \sigma, \mathtt{TempID}) = 1] = 1$$

where $(\sigma, \mathtt{TempID}, \mathtt{Adr}_{\mathtt{dst}}) \leftarrow \mathsf{SendRequest}(gpk, sk, \mathtt{TempID}, \mathtt{Adr}_{\mathtt{src}}, \mathtt{Adr}_{\mathtt{dst}}, \mathtt{Adr}_{\mathtt{proxy}})$, $(\sigma, \mathtt{TempID}, \mathtt{Adr}_{\mathtt{proxy}}) \leftarrow \mathsf{RelayRequest}(\mathtt{Adr}_{\mathtt{src}}, \mathtt{Adr}_{\mathtt{dst}}, \mathtt{Tbl}, \sigma, \mathtt{TempID})$, *and* $C \leftarrow \mathsf{SendContent}(gpk, \sigma, \mathtt{TempID}, M, \mathtt{Adr}_{\mathtt{proxy}})$.

Next, we define anonymity, semantic security, and unforgeability as follows. One session is defined as sequences of algorithm executions from SendRequest to GetContent, where SendRequest → RelayRequest → SendContent → RelayContent → GetContent. Anonymity guarantees that no adversary $\mathcal{A}$ who is allowed to communicate with the proxy (but is not allowed to know $\mathtt{Adr}_{\mathtt{src}}$) can distinguish whether the users of two different sessions are the same or not. In this game, $\mathcal{A}$ is modeled as a malicious SP. Moreover, we care about signing key exposure, where $\mathcal{A}$ can obtain signing keys. In addition to this, we give msk to $\mathcal{A}$ in order to guarantee that the KGC ability has nothing to right for identifying the user[5].

Definition 3.3 (Anonymity).

1. *The challenger $\mathcal{C}$ runs* $(gpk, ik) \leftarrow \mathsf{GM.Setup}(1^\lambda)$ *and* $(params, msk) \leftarrow \mathsf{KGC.Setup}(1^\lambda)$*, and computes two signing keys $sk_0, sk_1 \leftarrow \mathsf{Join}(gpk, ik)$, and gives gpk, sk_0, sk_1, and $(params, msk)$ to an adversary $\mathcal{A}$. Moreover, $\mathcal{C}$ initializes* $\mathtt{Tbl} := \emptyset$.
2. *$\mathcal{A}$ is allowed to issue the* SendRequest *query* $(i, \mathtt{TempID}) \in \{0,1\} \times \mathcal{ID}$*. $\mathcal{C}$ runs* $\mathsf{SendRequest}(gpk, sk_b, \mathtt{TempID}, \mathtt{Adr}_{\mathtt{src}}, \mathtt{Adr}_{\mathtt{dst}}, \mathtt{Adr}_{\mathtt{proxy}})$*, and returns σ (generated via the* SendRequest *algorithm) to $\mathcal{A}$.*
3. *$\mathcal{A}$ is allowed to issue the* RelayRequest *query* $(\sigma, \mathtt{TempID})$*. $\mathcal{C}$ runs* $\mathsf{RelayRequest}(\mathtt{Adr}_{\mathtt{src}}, \mathtt{Adr}_{\mathtt{dst}}, \mathtt{Tbl}, \sigma, \mathtt{TempID})$ *and updates* $\mathtt{Tbl}$.
4. *$\mathcal{A}$ is allowed to issue the* RelayContent *query C. $\mathcal{C}$ runs* $\mathsf{RelayContent}(C, \mathtt{Tbl})$*, and updates* $\mathtt{Tbl}$.
5. *$\mathcal{A}$ sends* $\mathtt{TempID}^* \in \mathcal{ID}$ *to $\mathcal{C}$. $\mathcal{C}$ flips a coin* $b \xleftarrow{\$} \{0,1\}$*, and runs* $(\sigma^*, \mathtt{TempID}^*, \mathtt{Adr}_{\mathtt{dst}}) \leftarrow \mathsf{SendRequest}(gpk, sk_b, \mathtt{TempID}^*, \mathtt{Adr}_{\mathtt{src}}, \mathtt{Adr}_{\mathtt{dst}}, \mathtt{Adr}_{\mathtt{proxy}})$ *and* $(\sigma^*, \mathtt{TempID}^*, \mathtt{Adr}_{\mathtt{proxy}}) \leftarrow \mathsf{RelayRequest}(\mathtt{Adr}_{\mathtt{src}}, \mathtt{Adr}_{\mathtt{dst}}, \mathtt{Tbl}, \sigma^*, \mathtt{TempID}^*)$*. $\mathcal{A}$ returns an arbitrary C to $\mathcal{C}$. $\mathcal{C}$ runs* $C \leftarrow \mathsf{RelayContent}(C, \mathtt{Tbl})$*. Note that $\mathcal{A}$ can know the transcript of these algorithms executed by $\mathcal{C}$:* $(\sigma^*, \mathtt{TempID}^*, \mathtt{Adr}_{\mathtt{dst}})$, $(\sigma^*, \mathtt{TempID}^*, \mathtt{Adr}_{\mathtt{proxy}})$*, and C. $\mathcal{A}$ outputs* $b' \in \{0,1\}$.

The protocol is said to have anonymity if $\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{pro},\mathcal{A}}(\lambda) := |\Pr[b = b'] - 1/2|$ *is negligible in λ.*

Next, we define semantic security which guarantees that no information of content M is revealed from the transcripts of algorithms. In this game, an adversary $\mathcal{A}$ is modeled as a malicious proxy. Moreover, $\mathcal{A}$ is allowed to obtain ik in order to guarantee that no information of M is revealed even from the GM's viewpoint.

Definition 3.4 (Semantic Security).

1. *The challenger $\mathcal{C}$ runs* $(gpk, ik) \leftarrow \mathsf{GM.Setup}(1^\lambda)$ *and* $(params, msk) \leftarrow \mathsf{KGC.Setup}(1^\lambda)$*, and gives gpk, ik, and $params$ to an adversary $\mathcal{A}$.*
2. *$\mathcal{A}$ is allowed to issue the* UserKeyGen *query* $\mathtt{TempID} \in \mathcal{ID}$*. $\mathcal{C}$ runs* $\mathsf{UserKeyGen}(params, msk, \mathtt{TempID})$ *and returns* $dk_{\mathtt{TempID}}$.
3. *$\mathcal{A}$ sends* $\mathtt{TempID}^* \in \mathcal{ID}, M_0^*, M_1^* \in \mathcal{M}$ *and sk^* to $\mathcal{C}$ as his choice, where* $\mathtt{TempID}^*$ *has not been sent as a* UserKeyGen *query. $\mathcal{C}$ flips a coin* $b \xleftarrow{\$} \{0,1\}$*, runs* $\mathsf{SendRequest}(gpk, sk^*, \mathtt{TempID}^*, \mathtt{Adr}_{\mathtt{src}}, \mathtt{Adr}_{\mathtt{dst}}, \mathtt{Adr}_{\mathtt{proxy}})$ *and* $C^* \leftarrow \mathsf{SendContent}(gpk, \sigma, \mathtt{TempID}, M_b^*, \mathtt{Adr}_{\mathtt{proxy}})$*, and sends* $(\sigma, \mathtt{TempID}^*, \mathtt{Adr}_{\mathtt{dst}})$ *and C^* to $\mathcal{A}$.*
4. *$\mathcal{A}$ is allowed to issue the* UserKeyGen *query* $\mathtt{TempID} \in \mathcal{ID}$ *where* $\mathtt{TempID} \neq \mathtt{TempID}^*$*. $\mathcal{C}$ runs* $\mathsf{UserKeyGen}(params, msk, \mathtt{TempID})$ *and returns* $dk_{\mathtt{TempID}}$.
5. *Finally, $\mathcal{A}$ outputs* $b' \in \{0,1\}$.

The protocol is said to have semantic security if $\mathsf{Adv}^{\mathsf{ss}}_{\mathsf{pro},\mathcal{A}}(\lambda) := |\Pr[b = b'] - 1/2|$ *is negligible in λ.*

Finally, we define unforgeability which guarantees that no adversary $\mathcal{A}$ who does not have a signing key will be accepted by the ValidityCheck algorithm. In this game, $\mathcal{A}$ is modeled as a malicious user. Moreover, $\mathcal{A}$ is allowed to obtain msk in order to guarantee that nobody can be accepted by SP even by KGC.

Definition 3.5 (Unforgeability).

1. *The challenger $\mathcal{C}$ runs* $(gpk, ik) \leftarrow \mathsf{GM.Setup}(1^\lambda)$ *and* $(params, msk) \leftarrow \mathsf{KGC.Setup}(1^\lambda)$*, and gives gpk and $params$ to an adversary $\mathcal{A}$. Moreover, $\mathcal{C}$ initializes* $\mathcal{S} = \emptyset$.
2. *$\mathcal{A}$ is allowed to issue the* SendRequest *query* $(i, \mathtt{TempID})$*. If sk_i has not been generated, then $\mathcal{C}$ runs* $sk_i \leftarrow \mathsf{Join}(gpk, ik)$ *and preserves sk_i. $\mathcal{C}$ runs* $\mathsf{SendRequest}(gpk, sk_i, \mathtt{TempID}, \mathtt{Adr}_{\mathtt{src}}, \mathtt{Adr}_{\mathtt{dst}}, \mathtt{Adr}_{\mathtt{proxy}})$*, and sends σ to $\mathcal{A}$. Moreover, $\mathcal{C}$ appends* $(\sigma, \mathtt{TempID})$ *to $\mathcal{S}$.*
3. *Finally, $\mathcal{A}$ outputs* $(\sigma^*, \mathtt{TempID}^*)$*. We say that $\mathcal{A}$ wins if* $(\sigma^*, \mathtt{TempID}^*) \notin \mathcal{S}$ *and* $\mathsf{ValidityCheck}(gpk, \sigma^*, \mathtt{TempID}^*) = 1$.

[4]For example, port numbers can be used for identifying the sessions. It's up to the proxy in our implementation.

[5]Note that we exclude the trivial case that KGC is offered to generate a decryption key of TempID from a user whose IP address is $\mathtt{Adr}_{\mathtt{src}}$, and observes that the transcript containing TempID.

The protocol is said to have unforgeability if $\mathsf{Adv}^{\mathsf{uf}}_{\mathsf{pro},\mathcal{A}}(\lambda) := \Pr[\mathcal{A}\ wins]$ *is negligible in* λ.

We say that a protocol is called secure anonymous authentication protocol if the protocol is correct and has anonymity, semantic security, and unforgeability.

3.2 Protocol Construction

Here, we give our proposed protocol construction. First, we define the syntax of building blocks - IBE and open-free group signature - as follows:

An IBE scheme $\mathcal{IBE}$ consists of four algorithms: i.e., (IBE.Setup, Extract, IBE.Enc, IBE.Dec). Let $\mathcal{ID}$ and $\mathcal{M}$ be an identity space and message space, respectively.

Definition 3.6 (Syntax of IBE [9]).

IBE.Setup: *This algorithm takes as input the security parameter* λ*, and outputs a public key params and a master secret key* msk.

Extract: *This algorithm takes as input params,* msk*, and an identity* $ID \in \mathcal{ID}$*, and outputs a decryption key* dk_{ID}.

IBE.Enc: *This algorithm takes as input params,* ID*, and a message* $M \in \mathcal{M}$*, and outputs a ciphertext* C_{IBE}.

IBE.Dec: *This algorithm takes as input params,* C_{IBE}*, and* dk_{ID}*, and outputs* M.

We require the following correctness property: for all $(params, msk) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$, all ID and all M, $\Pr[\mathsf{IBE.Dec}(params, \mathsf{IBE.Enc}(params, ID, M), \mathsf{Extract}(params, msk, ID)) = M] = 1$ holds.

An open-free group signature scheme $\mathcal{GS}$ consists of four algorithms: (GS.Setup, Join, Sign, Verify)[6] as follows:

Definition 3.7 (Syntax of Open-Free Group Signature).

GS.Setup: *This algorithm takes as input the security parameter* λ*, and outputs a group public key gpk and an issuer key ik.*

GS.Join: *This algorithm takes as input gpk and ik (from GM), and a user is obtained a signing key sk.*

Sign: *This algorithm takes as input gpk, a signing key sk, and a message* M*, and outputs a group signature* σ.

Verify: *This algorithm takes as input gpk,* σ*, and* M*, and outputs* 1 *if* σ *is a valid signature on* M*, and 0 otherwise.*

We require the following correctness property: for all $(gpk, ik) \leftarrow \mathsf{GS.Setup}(1^\lambda)$ and $sk \leftarrow \mathsf{GS.Join}(gpk, ik)$, $\Pr[\mathsf{Verify}(gpk, \mathsf{Sign}(gpk, sk, M), M) = 1] = 1$ holds.

Next, we give our proposed construction. In this construction, a signed message of the underlying group signature is TempID which is also regarded as a public key of the underlying IBE.

[6]This new primitive is a kind of dynamic group signature, where a new member can join the system even after the setup phase. Note that, additional two algorithms, Open and Judge, are usually contained in dynamic group signatures (e.g. [7]). The Judge algorithm checks a proof output by the Open algorithm, whether the Open algorithm is correctly executed or not. Obviously, the Judge algorithm is meaningless in the open-free variant.

Construction 3.1 (Proposed Protocol).

GM.Setup : *Run* $(gpk, ik) \leftarrow \mathsf{GS.Setup}(1^\lambda)$*, and output* (gpk, ik).

KGC.Setup : *Run* $(params, msk) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$*, and output* $(params, msk)$.

Join : *Run* $sk \leftarrow \mathsf{GS.Join}(gpk, ik)$*, and output* sk.

UserKeyGen : *Run* $dk_{\mathtt{TempID}} \leftarrow \mathsf{Extract}(params, msk, \mathtt{TempID})$*, and output* $dk_{\mathtt{TempID}}$.

SendRequest : *Choose* $\mathtt{TempID} \xleftarrow{\$} \mathcal{ID}$*. Run* $\sigma \leftarrow \mathsf{Sign}(gpk, sk, \mathtt{TempID})$*, and send* $(\sigma, \mathtt{TempID}, \mathtt{Adr}_{\mathtt{dst}})$ *to the proxy whose IP address is* $\mathtt{Adr}_{\mathtt{proxy}}$.

RelayRequest : *Append* $(\mathtt{TempID}, \mathtt{Adr}_{\mathtt{src}})$ *to* Tbl*, and relays a pair* $(\sigma, \mathtt{TempID})$ *and* $\mathtt{Adr}_{\mathtt{proxy}}$ *to the destination SP whose IP address is* $\mathtt{Adr}_{\mathtt{dst}}$.

ValidityCheck : *Output* 1 *if* $\mathsf{Verify}(gpk, \sigma, \mathtt{TempID}) = 1$*, and* 0*, otherwise.*

SendContent : *Output* $\perp$ *if* $\mathsf{ValidityCheck}(gpk, \sigma, \mathtt{TempID}) = 0$*. Otherwise, run* $C_{IBE} \leftarrow \mathsf{IBE.Enc}(params, \mathtt{TempID}, M)$*, and send* C_{IBE} *to the proxy whose IP address is* $\mathtt{Adr}_{\mathtt{proxy}}$.

RelayContent : *Relay* C_{IBE} *to a user whose IP address is* $\mathtt{Adr}_{\mathtt{src}}$ *contained in* Tbl*. Moreover, remove* $(\mathtt{TempID}, \mathtt{Adr}_{\mathtt{src}})$ *from* Tbl.

GetContent : *Output the result of* $\mathsf{IBE.Dec}(params, C_{IBE}, dk_{\mathtt{TempID}})$.

Note that our the above construction only considers one-proxy setting, and therefore no anonymity is guaranteed from the viewpoint of the Proxy, since the Proxy directly relays communications between the User and SP. Note that this situation does not contradict our definition of anonymity (Def. 3.3). We can simply extend this protocol to a multi-proxy setting, where each Proxy relays $(\sigma, \mathtt{TempID})$ or C_{IBE} between the previous Proxy and the next Proxy. Then, anonymity is guaranteed even from the Proxies' point of view unless all Proxies collude with each other.

4. GROUP SIGNATURE

The proposed secure anonymous authentication protocol uses a group signature scheme as its fundamental component. Though arbitrary group signature schemes could be used (i.e., by ignoring open functionality), it is beneficial to remove unnecessary functionality and improve performance efficiency, thus the proposed protocol in Section 3.2 uses a group signature without open functionality. We call this an open-free group signature[7]. This section defines the security of such signatures.

[7]A difference between ring signature [20] and open-free group signature is as follows. In ring signature schemes, a signer chooses a set of other members, and signs on behalf of the group of users. The anonymity of the signer cannot be revoked in contrast to group signature schemes. However, a signer needs to involve/choose other members when the signer signs, and therefore needs to know other members. This does not match our setting.

4.1 Defining Open-free Group Signature

In this section, we redefine the security definitions of the Furukawa-Imai group signature scheme, anonymity, traceability, and non-frameability, to match the open-free variant. Anonymity guarantees that no adversary $\mathcal{A}$ can distinguish whether two signers of group signatures are the same or not, even if $\mathcal{A}$ has the corresponding signing keys. Usually, there are two kind of anonymity, CPA-anonymity and CCA-anonymity. In CCA-anonymity, $\mathcal{A}$ is allowed to issue open queries, where $\mathcal{A}$ sends (σ, M), and is given the result of the Open algorithm. Meanwhile, we do not have to consider these differences due to the open-free property.

Definition 4.1 (Anonymity).

1. *An adversary $\mathcal{A}$ with the security parameter λ sends gpk, sk_0, sk_1, and M to the challenger $\mathcal{C}$.*
2. *$\mathcal{C}$ chooses $b \xleftarrow{\$} \{0,1\}$, computes $\sigma^* \leftarrow \mathsf{Sign}(gpk, sk_b, M)$, and sends σ^* to $\mathcal{A}$.*
3. *$\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.*

An open-free group signature $\mathcal{GS}$ is said to have anonymity if $\mathsf{Adv}^{\mathsf{anon}}_{\mathcal{GS},\mathcal{A}}(\lambda) := |\Pr[b = b'] - 1/2|$ is negligible in λ.

Next, we redefine traceability. In usual definition, Traceability guarantees that no adversary $\mathcal{A}$ can produce a valid-but-untraceable group signature, that is, the Open algorithm cannot identify the corresponding signer though the Verify algorithm outputs 1. However, in the open-free variant, this definition is meaningless. So, we define unforgeability here instead of traceability, where no adversary $\mathcal{A}$ can produce a valid group signature without knowing a signing key.

Definition 4.2 (Unforgeability).

1. *The challenger $\mathcal{C}$ runs $(gpk, ik) \leftarrow \mathsf{GS.Setup}(1^\lambda)$, and gives gpk to an adversary $\mathcal{A}$.*
2. *$\mathcal{A}$ is allowed to issue the signing query (M, i). If a user U_i has not been joined to the system, then $\mathcal{C}$ runs the $\mathsf{GS.Join}$ algorithm, computes sk_i, and returns $\sigma \leftarrow \mathsf{Sign}(gpk, sk_i, M)$ to $\mathcal{A}$. If U_i has been joined to the system, then $\mathcal{C}$ returns $\sigma \leftarrow \mathsf{Sign}(gpk, sk_i, M)$ to $\mathcal{A}$. Moreover, $\mathcal{C}$ appends (σ, M) into the list $\mathcal{S}$.*
3. *Finally, $\mathcal{A}$ outputs (σ^*, M^*). We say that $\mathcal{A}$ wins if $\mathsf{Verify}(gpk, \sigma^*, M^*) = 1$ holds and $(\sigma^*, M^*) \notin \mathcal{S}$.*

An open-free group signature $\mathcal{GS}$ is said to have unforgeability if $\mathsf{Adv}^{\mathsf{un}}_{\mathcal{GS},\mathcal{A}}(\lambda) := \Pr[\mathcal{A}\ wins]$ is negligible in λ.

Finally, we revisit non-frameability. Non-frameability guarantees that no adversary $\mathcal{A}$ can produce a valid group signature whose open result is an honest (i.e., uncorrupted by $\mathcal{A}$) user (say U). Obviously, this definition is meaningless in the open-free variant, and therefore we do not consider non-frameability.

Note that in order to achieve non-frameability in the original scheme, a user chooses a secret key usk, and is obtained its signing key sk by executing the $\mathsf{GS.Join}$ algorithm with GM. What is critical, GM cannot know usk itself (but can convince that the user knows usk by using zero-knowledge proofs). In other words, we can remove a secret key usk from the syntax of group signature unless non-frameability is required. This is is the reason why we do not require any secret key of users as input of the $\mathsf{GS.Join}$ algorithm, and the $\mathsf{GS.Join}$ algorithm can be a non-interactive algorithm.

4.2 Building Open-Free Group Signature

Our group signature scheme modifies the Furukawa-Imai group signature [13]. In the Furukawa-Imai scheme, a user certificate issued by the GM is a short signature [8]. The user proves the possession of the certificate by NIZK proofs which are constructed via the Fiat-Shamir conversion [12]. For implementing the Open algorithm, an ElGamal-type double encryption is used over a decisional Diffie-Hellman (DDH)-hard group (in addition to bilinear groups). In our open-free scheme, the DDH-hard group can be removed. Other part is the same as that of the original Furukawa-Imai group signature scheme.

Note that, a simple construction, where for one signature verification/signing key pair $(\mathsf{VK}, \mathsf{SK})$, each group member shares SK, can also be seen as an open-free group signature scheme. However, this simple construction never realizes the revocation functionality [17]. Towards constructing a revocable open-free group signature scheme, we newly construct an open-free group signature scheme.

Construction 4.1 (Proposed Open-Free Group Signature).

$\mathsf{GS.Setup}$: *Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be a bilinear group with prime order p, where $\langle g_1 \rangle = \mathbb{G}_1$, $\langle g_2 \rangle = \mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map[8]. Choose $\gamma \xleftarrow{\$} \mathbb{Z}_p$, and $h \xleftarrow{\$} \mathbb{G}_1$, and compute $W = g_2^\gamma$. Output $gpk = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, h$ and $ik = \gamma$, where $H_3 : \{0,1\}^* \to \mathbb{Z}_p$ is a hash function modeled as a random oracle.*

$\mathsf{GS.Join}$: *For a user U_i, choose $x_i, y_i \xleftarrow{\$} \mathbb{Z}_p$, compute $A_i = (g_1 h^{-y_i})^{\frac{1}{\gamma + x_i}}$, and output $sk_i = (x_i, y_i, A_i)$.*

Sign: *Let $sk = (x, y, A)$. Choose $\beta \xleftarrow{\$} \mathbb{Z}_p$, set $\delta = \beta x - y$, and compute $T = Ah^\beta$. Choose $r_x, r_\delta, r_\beta \xleftarrow{\$} \mathbb{Z}_p$, and compute $R = e(h, g_2)^{r_\delta} e(h, W)^{r_\beta} / e(T, g_2)^{r_x}$, $c = H_3(gpk, T, R, M)$, $s_x = r_x + cx$, $s_\delta = r_\delta + c\delta$, and $s_\beta = r_\beta + c\beta$, and output $\sigma = (T, c, s_x, s_\delta, s_\beta)$.*

Verify: *Compute $R' = \frac{e(h,g_2)^{s_\delta} e(h,W)^{s_\beta}}{e(T,g_2)^{s_x}} \left(\frac{e(T,W)}{e(g_1,g_2)}\right)^{-c}$, and output 1 if $c = H_3(gpk, T, R', M)$ holds, and 0 otherwise.*

Compared to the original Furukawa-Imai scheme, we can reduce three DDH-hard group elements and three $\mathbb{Z}_p$ elements. Accordingly, we can reduce the size of signature by 50% compared to the original Furukawa-Imai group signature scheme.

5. IMPLEMENTATION AND DISCUSSION

5.1 Analysis on Proposed Protocol

We can prove that our proposed protocol is secure if the underlying IBE scheme is IND-ID-CPA secure (like the Boneh-Franklin IBE scheme [9]) and the underlying group signature scheme is anonymous and unforgeable. We only give a sketch of proof of anonymity (other theorems can be similarly proved) here and omit the full proofs of the following theorems due to the page limitation.

Theorem 5.1. *Our protocol is anonymous if the underlying group signature scheme is anonymous.*

[8]We require bilinearity: for all $a, b \in \mathbb{Z}_p$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab} = e(g_1^b, g_2^a)$, and non-degeneracy: $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$, where $1_{\mathbb{G}_T}$ is the identity element in $\mathbb{G}_T$.

PROOF (SKETCH). Let $\mathcal{A}$ be an adversary who can break anonymity of our protocol. Then, we can construct an algorithm $\mathcal{B}$ that breaks anonymity of the underlying group signature scheme as follow. Let $\mathcal{C}$ be the challenger of the underlying group signature. $\mathcal{B}$ generates gpk, sk_0, and sk_1, and generates all IBE-related values. Then $\mathcal{B}$ gives $(gpk, sk_0, sk_1, params, msk)$ to $\mathcal{A}$. In the challenge phase, $\mathcal{B}$ gets $\texttt{TempID}^*$ from $\mathcal{A}$, forwards it to $\mathcal{C}$, and gets σ^* from $\mathcal{C}$. $\mathcal{B}$ uses σ^* as the output of the SendRequest algorithm, and similarly simulates other algorithms. $\mathcal{A}$ outputs b' and $\mathcal{B}$ also outputs b' as the guessing bit. Then, $\mathcal{B}$ can break anonymity of the group signature with the same advantage of $\mathcal{A}$. This contradicts that the underlying group signature is anonymous. □

THEOREM 5.2. *Our protocol is semantic secure if the underlying IBE scheme is IND-ID-CPA secure.*

THEOREM 5.3. *Our protocol is unforgeable if the underlying group signature scheme is unforgeable.*

5.2 Analysis on Group Signature

The remaining part is to show that the proposed open-free group signature scheme is anonymous and unforgeable. The proposed open-free group signature scheme is constructed from an (honest-verifier) zero-knowledge proof of knowledge by using the Fiat-Shamir conversion [12]. First, we explain the original proof of knowledge protocol as follows. A prover computes (T, R), and sends it to a verifier. The verifier sends a challenge value c to the prover. The prover computes (s_x, s_δ, s_β), and sends it to the verifier. The verifier checks whether the verification equation holds or not. Next, we show that this 3-move protocol is zero-knowledge (this immediately leads to anonymity). The simulator chooses $A \xleftarrow{\$} \mathbb{G}$ and $\beta \xleftarrow{\$} \mathbb{Z}_p$, and computes $T = Ag_1^\beta$. Note that β is chosen uniformly random. Therefore, T generated from the simulator is drawn from a distribution that is indistinguishable from the distribution output by any particular prover. For $T \in \mathbb{G}$, the simulator chooses $c, s_x, s_\delta, s_\beta \xleftarrow{\$} \mathbb{Z}_p$, and computes $R = \frac{e(h,g_2)^{s_\delta} e(h,W)^{s_\beta}}{e(T,g_2)^{s_x}} \left(\frac{e(T,W)}{e(g_1,g_2)}\right)^{-c}$. Then the transcript $(T, R, c, s_x, s_\delta, s_\beta)$ here is indistinguishable from transcripts of the actual protocol.

Next, we show that the protocol is a proof of knowledge. That is, we show there exists an extractor that can extract a SDH pair from $(T, R, c, s_x, s_\delta, s_\beta)$ and $(T, R, c', s'_x, s'_\delta, s'_\beta)$, where $c \neq c'$ and both transcripts satisfy the verification equation. Set $\tilde{x} := \frac{s_x - s'_x}{c - c'}$, $\tilde{y} := \frac{(s_x - s'_x)(s_\beta - s'_\beta) - (s_\delta - s'_\delta)(c - c')}{(c - c')^2}$, and $\tilde{\beta} := \frac{s_\beta - s'_\beta}{c - c'}$. Then, $\frac{e(T,W)}{e(g_1,g_2)} = \frac{e(h,g_2)^{\tilde{\beta}\tilde{x} - \tilde{y}} e(h,W)^{\tilde{\beta}}}{e(T,g_2)^{\tilde{x}}}$ holds. Therefore, for $\tilde{A} = T/h^{\tilde{\beta}}$, $e(\tilde{A}, g^{\tilde{x}}W) = e(g_1, g_2)e(h, g_2)^{-\tilde{y}}$ holds. That is, $(\tilde{x}, \tilde{y}, \tilde{A})$ can be extracted. This immediately leads to unforgeability. We omit the formal proof since this is similar as that of the original Furukawa-Imai scheme.

5.3 Prototype

This section introduces a prototype that implements the proposed protocol and evaluates its performance to demonstrate the feasibility and practicality of the protocol.

5.3.1 Implementation

We built the User and SP modules by using C language (GCC version 4.2.1). We also used the TEPLA library [4] for implementing the Boneh-Franklin IBE scheme and our open-free group signature scheme. This library supports optimal Ate pairings over Barreto-Naehrig (BN) elliptic curves [6] with 254-bit prime order and the corresponding embedded degree is 12. This enables 128-bit security. We used Simpleproxy [3] for the Proxy module.

Three types of communication sequences are implemented, i.e., User-GM, User-KGC, and User-Proxy-SP, and each of the sequence runs the modules defined in Section 3.2. The User-GM sequence begins with the Join module, which communicates with the GM. The GM then computes the signing key sk, and returns it to the User. The User-KGC sequence begins with the UserKeyGen module, which communicates with the KGC. The User sends TempID to the KGC, and the KGC then computes the decryption key $dk_{\texttt{TempID}}$, and returns it to the User. The User-Proxy-SP sequence begins with the SendRequest module that sends a group signature and TempID. Upon receiving them, the Proxy runs RelayRequest module that forwards them to the SP. It then runs ValidityCheck module and SendContent module that returns an IBE ciphertext to the Proxy, which forwards that to the User. The User then runs GetContent module that decrypts the IBE ciphertext by using the corresponding $dk_{\texttt{TempID}}$.

Note that the User-GM sequence needs to be run before User-Proxy-SP sequence starts. Likewise, the User-KGC and User-Proxy-SP sequences are run in parallel, though the User-KGC procedure needs to be completed before User-Proxy-SP procedure's GetContent module is run.

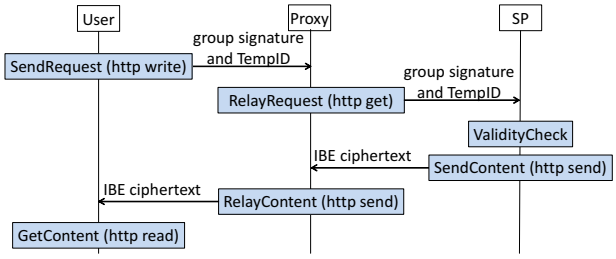

Figure 2: Sequences for User-Proxy-SP

5.3.2 Performance Measurement

An environment for performance evaluation of the proposed protocol was prepared. We used an Apple MacBookPro 15inch mid 2010 (processor: 2.8GHz Intel Core i7, Memory: 8GB, 1067 MHz DDR3, Darwin Kernel Version 12.4.0), and prepared two VMs by using VMware Fusion 5.0.3. We assigned the roles of the User to the MacOS the roles of the Proxy and SP on the VMs. For the Proxy, the VM ran FreeBSD amd64 9.1-RELEASE with one processor and 256MB of memory, and for the SP, VM ran CentOS 5.9 x86_64 with one processor and 512MB of memory.

Here, we show that our protocol is feasible by showing the running time of algorithms and total running time of one session are msec order. First, we show the running time of one session (User→Proxy→SP→Proxy→User) in the following cases: (1) HTTP communications (i.e., without any cryptographic operations), (2) SSL communications, and (3) our protocol in Table 1. To measure the running time of the SSL communication, we use the s_server/s_time command of the OpenSSL library (ver. 1.0.1e) [1]. We use DHE-RSA-AES128-SHA256 cipher suite with a 3072-bit size public key since this also supports 128-bit security, as in ours.

Table 1 shows that the running time of our protocol is approximately 50-times slower than that of SSL communications. This inefficiency is due to the pairing computation

Table 1: Running Time (one session)

Scheme	Time(msec)	Cryptographic Operations
None	4.714	-
SSL	12.897	Enc/Auth
Ours	624.743	Enc/Anon. Auth

which is not required in usual public key encryption, digital signature, and authentication (these are used in SSL). Nevertheless, it is particularly worth noting that our running time still fits inside millisecond order, and our protocol even supports secure, anonymous, and authenticated communication, simultaneously.

For reference, Table 2 gives the running time of each algorithm. Note that the GM.Setup, KGC.Setup, and Join algorithms can be run offline, and the UserKeyGen algorithm can be run separately against the session. Moreover, we ignore the RelayRequest and RelayContent algorithms since these (run by Proxy) just relay the communication, and are run independently against any cryptographic operations.

Table 2: Running Time (algorithms)

Algorithm	Time(msec)	Entity
GM.Setup	105.712	GM
KGC.Setup	102.883	KGC
Join	109.036	User-GM
UserKeyGen	102.958	User-KGC
SendRequest	125.069	User
ValidityCheck	199.247	SP
SendContent	198.636	SP
GetContent	101.158	User

The dominant factor for User is the SendRequest algorithm which computes a group signature. Note that this procedure can also be run offline by assuming that the User chooses `TempID` and computes a group signature before starting a session. Then, the total running time of one session becomes less than 500 msec.

6. CONCLUSION

The proposed protocol along with our group signature enables secure anonymous authentication. It is feasible and practical in terms of transaction time. Although this paper proved its concept, we need to consider practical deployment over the Internet. Indeed, the protocol requires a proxy that assists secure, anonymous, and authenticated communication. Though various types of proxy may exist, including Tor routers, we need to consider and verify the adaptability of our protocol to the current infrastructure. On the other hand, assorted anonymous communication systems [5, 15, 19] have risks of being used by malicious parties. One reason for that is their inability to authenticate users. Properly applying our protocol may enable these systems to be properly used. Through this work, we wish to facilitate secure, anonymous, and authenticated communication over the Internet.

Acknowledgment: The authors would like to thank Dr. Goichiro Hanaoka and Dr. Miyako Ohkubo for their invaluable comments.

7. REFERENCES

[1] OpenSSL: Cryptography and SSL/TLS Toolkit. Available at `http://www.openssl.org/`.
[2] Selected Papers in Anonymity. Available at http://freehaven.net/anonbib/date.html.
[3] Simpleproxy: Crocodile group software. Available at `http://www.crocodile.org/software.html`.
[4] TEPLA: University of Tsukuba Elliptic Curve and Pairing Library. Available at `http://www.cipher.risk.tsukuba.ac.jp/tepla/index_e.html`.
[5] Tor Project. Available at https://www.torproject.org/.
[6] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography*, pages 319–331, 2005.
[7] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA*, pages 136–153, 2005.
[8] D. Boneh and X. Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008.
[9] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.
[10] D. Chaum and E. van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.
[11] J. H. Cheon, N. Hopper, Y. Kim, and I. Osipkov. Provably secure timed-release public key encryption. *ACM Trans. Inf. Syst. Secur.*, 11(2), 2008.
[12] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.
[13] J. Furukawa and H. Imai. An efficient group signature scheme from bilinear maps. *IEICE Transactions*, 89-A(5):1328–1338, 2006.
[14] Y. Gilad and A. Herzberg. Plug-and-play IP security - anonymity infrastructure instead of PKI. In *ESORICS*, pages 255–272, 2013.
[15] A. Houmansadr, C. Brubaker, and V. Shmatikov. The parrot is dead: Observing unobservable network communications. In *IEEE S&P*, pages 65–79, 2013.
[16] M. Z. Lee, A. M. Dunn, B. Waters, E. Witchel, and J. Katz. Anon-pass: Practical anonymous subscriptions. In *IEEE S&P*, pages 319–333, 2013.
[17] B. Libert, T. Peters, and M. Yung. Group signatures with almost-for-free revocation. In *CRYPTO*, pages 571–589, 2012.
[18] B. Libert and D. Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. *IEEE Trans. on Information Theory*, 57(3):1786–1802, 2011.
[19] H. M. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg. SkypeMorph: protocol obfuscation for Tor bridges. In *ACM CCS*, pages 97–108, 2012.
[20] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *ASIACRYPT*, pages 552–565, 2001.
[21] A. Sudarsono, T. Nakanishi, Y. Nogami, and N. Funabiki. Anonymous IEEE802.1X authentication system using group signatures. *JIP*, 18:63–76, 2010.