ISDSR+: Improving the Security and Availability of Secure Routing Protocol

Hideharu Kojima, Naoto Yanai, Jason Paul Cruz

Graduate School of Information Science and Technology, Osaka University
1-5, Yamadaoka, Suita, Japan

Abstract

In ad hoc networks that allow devices to dynamically configure networks via wireless communication, a secure routing protocol is a technology that guarantees the validity of routing with the use of cryptographic authentication. The secure DSR with ID-based sequential aggregate signatures (ISDSR) is a recently introduced secure routing protocol with "cryptographically compact chain," where each device signs both the routing information and signatures generated by the previous device without increasing the size of signatures and the ID information propagated via packets can be utilized as a public key. However, ISDSR requires communication with a centralized key generation center (KGC), and thus a new device may experience difficulties joining the protocol. Moreover, implementation results of ISDSR have not been presented. In this work, we present ISDSR+, a new secure routing protocol without a centralized KGC that uses distributed key generation and the conventional features of ISDSR. ISDSR+ relies on a novel signature scheme where any node can receive a secret key as long as the number of available KGCs is more than a certain threshold. In other words, even if several KGCs become unavailable, the remaining available KGCs can still provide secret keys. We furthermore show promising results of ISDSR+ via a prototype implementation on Raspberry Pis. The results show calculation time of 0.1 seconds for both the secret key generation and the round trip time (RTT) of routing information under reasonable settings. The RTT of ISDSR+ is better than that of a naive secure routing protocol with RSA signatures.

Keywords: Ad Hoc Networks, Secure Routing Protocol, Distributed Key Generation, ISDSR, Implementation

1. Introduction

1.1. Background

In mobile ad hoc networks, devices such as smartphones and sensors dynamically configure the networks in wireless communication without any fixed infrastructure. Therefore, mobile ad hoc networks offer many applications for important and emergency situations, e.g., infrastructure monitoring systems, such as railways (Velagandula et al., 2017), land slide detection (Giorgetti et al., 2016), and underground coals (Li and Liu, 2008), and emergency communication systems during disasters (Kato, 2014). To communicate with a destination device outside its direct transmission area in an ad hoc network, a device exchanges routing information with other devices via a routing protocol, such as AODV (Perkins et al., 2003a) and DSR (Johnson et al., 2007a), and the packets are forwarded until they reach the destination device. In several protocols (Perkins et al., 2003b; Johnson et al., 2007b), a routing information is a concatenation of device identifiers, such as IP address or device name, and a routing protocol requires each device to include its own identifier in the routing information it receives.

Despite its attractive advantages, an ad hoc network typically has weak security because it does not have a fixed infrastructure and it uses dynamic network configuration. For example, an adversary can forward packets along another direction that does not lead to the correct destination or loop packets infinitely by manipulating routing information or injecting fake information. These threats are based on an attack against a routing protocol and can often cause serious problems (Karlof and Wagner, 2003). Such attack can, for example, prevent the communication of a disaster-stricken user who wants to send an emergency call as soon as possible.

Secure Routing Protocols. Secure routing protocols (Zhou and Haas, 1999; Hu et al., 2002; Zapata and Asokan, 2002) prevent the attack described above by guaranteeing the validity of routing information with the use of cryptographic authentication schemes, such as digital signatures. The first secure routing protocol (Zhou and Haas, 1999) was based on a key management protocol, while subsequent works (Zapata and Asokan, 2002; Lee et al., 2003; Kim and Tsudik, 2009; Ács, 2009; Ghosh and Datta, 2011, 2013) adopted digital signatures with public verifiability and non-repudiation. When devices exchange routing information, each device generates a digital signature for the routing information and then sends both the signature and the routing

information. Then, another device that receives both the signature and the routing information can check the validity of these information by verifying the signature.

The main difficulty in constructing a secure routing protocol is guaranteeing both the *security and efficiency*, even in large-scale networks. In general, security and efficiency are traded off against each other. The validity of a whole routing information from a source to a destination should be guaranteed, but the introduction of an individual signature per node may decrease performance. Moreover, the security of a routing protocol may not be guaranteed when not all individual signatures from a source to a destination are sent. In several routing protocols (Hu et al., 2002; Zapata and Asokan, 2002), digital signatures were exchanged on a part of a whole routing information in networks, but these protocols were later shown to be insecure (May, 2009; Sanzgiri et al., 2005). However, trivially introducing all individual signatures along with routing information increases packet size in proportion to the number of nodes, and then a packet may be dropped because of its ballooning size. Therefore, the tradeoff between security and efficiency should be solved to design an adequate secure routing protocol.

Our Motivation. The *secure DSR with ID-based sequential aggregate signatures (ISDSR)* (Muranaka et al., 2016) has been proposed as a potential solution to the tradeoff problem described above. ISDSR can utilize ID information of each device as a public key, always keep a single short signature independent of the number of devices, and guarantee the whole routing information from any source to a destination. In comparison with other works (Zapata and Asokan, 2002; Lee et al., 2003; Song et al., 2005; Sanzgiri et al., 2005; Kim and Tsudik, 2009; Ács, 2009; Ghosh and Datta, 2011, 2013), ISDSR does not require either the linear size of signatures with respect to the number of hops or the use of public key certificates despite the inclusion of all signatures from a source to a destination. These advantages can be obtained with the use of a state-of-the-art cryptographic scheme named ID-based sequential aggregate signatures (Boldyreva et al., 2010).

However, ISDSR contains two disadvantages. **First, ISDSR requires a centralized key generation center (KGC) to generate a secret key for a requesting node**. Even though ISDSR does not have a fixed infrastructure, a KGC is essentially identical to a fixed infrastructure that a device needs to communicate with to receive a secret key. This kind of communication may become a serious problem particularly during disaster

situations, where a device of a user who wants to make an emergency call cannot join a protocol dynamically due to a loss of communication to the KGC. **Second, implementation results of ISDSR have never been presented**. The effects of decreasing the number of individual signatures and their related information on the performance of the protocol have not been verified. In addition, there are no results that show how the use of state-of-the-art cryptographic schemes improves the performance of the protocol. Therefore, unless these two weaknesses are solved, ISDSR remains incomplete as a secure routing protocol for ad hoc networks.

This work is a full version of our previous work (Kojima and Yanai, 2017) presented in CANDAR 2017. In the previous work, we implemented the signing algorithm of ISDSR in Java, which is closest to a realistic environment in terms of implementation on actual devices and on Android platforms. However, we did not discuss the problems of ISDSR described above. In this work, we revise the signing algorithm to remove the need for a centralized KGC and implement a prototype of the proposed protocol.

1.2. Contributions

In this work, we present *ISDSR+*, a new secure routing protocol without a centralized KGC, and show its performance via prototype implementation on Raspberry Pi. ISDSR+ is based on a novel signature scheme that allows devices to obtain secret keys as long as the number of available KGCs is more than a certain threshold. The key generation capability of ISDSR+ can be setup in an arbitrary setting, i.e., the number of KGCs and the threshold number for the signature scheme can be chosen arbitrarily. Therefore, ISDSR+ can be considered as a secure routing protocol without any fixed infrastructure. Then, based on implementation results of experiments on Raspberry Pi, we confirm that the computational time of ISDSR+ is faster than that of an RSA-based secure routing protocol, which is a naive-but-fast construction with liner communication costs. Our performance evaluations of ISDSR+ can be used as an indicator when an application on an ad hoc network adopts a secure routing protocol utilizing state-of-the-art cryptographic schemes. The contributions of this work are presented below.

ID-based sequential aggregate signatures with distributed key generation. The first contribution is the removal of a centralized KGC by extending the ID-based sequential aggregate signatures (Boldyreva et al., 2010) with the introduction of distributed key generation. A device can receive a secret key as

long as it can communicate with a number of KGCs more than a threshold designated in advance. In other words, even if several KGCs are unavailable, key generation can still be performed with the remaining accessible KGCs. We call this new scheme *ID-based sequential aggregate signatures with distributed key generation (IBSAS-DKG)*. We also show the security analysis of the digital signature algorithm with distributed key generation in a formal proof (See Section 4 for details).

Prototype Implementation. The second contribution is the evaluation of the performance of computational time as a round trip time (RTT) by implementing the signature algorithm of ISDSR+ in Java on Raspberry Pi. Our prototype utilizes the JPBC library[1] for the computation of elliptic curves and pairing functions. We also refactored our previous implementation (Kojima and Yanai, 2017) and the computational time of the signature algorithm of ISDSR+ is faster by more than 48 times. For our distributed key generation, in the 10-out-of-100 setting, a pair of partial secret key and master public key for a KGC and a secret key for a user can be computed in approximately 330 milliseconds and 100 milliseconds, respectively. Meanwhile, the ISDSR+ has an RTT of 0.1 seconds, which is faster than the RSA-based protocol with RTT of 0.24 seconds. These results show that ISDSR+ can be effectively used in ad hoc networks under reasonable settings (See Section 6 for details on our experiments and potential performance).

2. Related Works

In this section, we first describe related works about secure routing protocols. Then, we describe literature on digital signatures, distributed key generation, and security analysis of secure routing protocols.

Secure Routing Protocols. The works by (Kim and Tsudik, 2009; Ghosh and Datta, 2011; Muranaka et al., 2015) are the closest to our work in terms of the use of signature schemes for multiple signers, where the size of signatures is independent of the number of signers. The motivations of these works are to decrease communication overheads and guarantee the whole routing information. However, they need public key infrastructure while our protocol can achieve infra-less setting. To the best of our knowledge, the

[1] `http://gas.dia.unisa.it/projects/jpbc/`

first work that presented a secure routing protocol with an attractive cryptographic scheme was the paper in (Lee et al., 2003). This protocol and its subsequent work (Song et al., 2005) were based on ID-based cryptography (Shamir, 1984), which allows any user to utilize any string as a public key. Our protocol contains the advantages of these protocols in addition to the infra-less setting.

Digital Signatures. Among the existing digital signature schemes, schemes where signatures are combined into a single short signature without depending on the number of signers are called *multi-signature schemes* (Itakura and Nakamura, 1983). Multi-signature schemes are classified into two types, i.e., the interactive type (Boldyreva, 2003; Boneh et al., 2003; Gentry and Ramzan, 2006; Bellare and Neven, 2006), where signers interact with each other to generate a single signature, and the sequential type (Lysyanskaya et al., 2004; Boldyreva et al., 2010; Yanai et al., 2013), where signers generate a single signature from a signature-so-far without interaction. The sequential type is more suitable for secure routing protocols, which need signature chains to guarantee the validity of routing information received from nodes. ID-based schemes are more efficient than other schemes, and the sequential type in the ID-based cryptography, i.e., the scheme in (Boldyreva et al., 2010), is the most suitable scheme for secure routing protocols.

Distributed Key Generation. Distributed generation of secret keys is suitable for discrete-log-based cryptosystems (Desmedt and Frankel, 1989) and there are many such constructions (Gennaro et al., 1996, 1999; Park and Kurosawa, 1996). The construction of our scheme is close to the threshold signatures proposed by Boldyreva (Boldyreva, 2003).

Provable Security of Secure Routing Protocols. Buttyán and Vajda (Buttyán and Vajda, 2004) defined a formal security model of ad hoc networks and their routing protocols and analyzed the security of several secure routing protocols (Papadimitratos and Haas, 2002; Hu et al., 2002, 2005). The model by Buttyán and Vajda discussed the security against a single malicious node, and the security was extended by Ács (Ács, 2009) into a model with multiple malicious nodes. Their results show that a secure routing protocol is secure if the digital signature scheme utilized is secure. Kim and Tsudik (Kim and Tsudik, 2009) extended the results of Ács and presented a case utilizing a signature scheme for multiple signers. Our protocol-level security can be

guaranteed via the Kim-Tsudik framework (Kim and Tsudik, 2009) although we omit the details due to space limitation.

3. ISDSR+

In this section, we describe a system model of ISDSR+. As described in Section 1, ISDSR+ is an extension of the secure DSR with ID-based sequential aggregate signatures (ISDSR). In particular, we introduce distributed key generation in ISDSR+. ISDSR utilizes an ID-based sequential aggregate signature scheme (Boldyreva et al., 2010), where each user generates a single short signature by taking both messages to be signed and a signature-so-far as input and utilizes any string as its own public key to guarantee routing information sent via packets.

3.1. Protocol Overview

ISDSR+ consists of *distributed key generation*, *secure route discovery*, and *secure route maintenance*. Hereinafter, we assume that each node utilizes its own ID information, e.g., IP address or device name, as a public key. In the distributed key generation phase, each node first broadcasts *key request packets (KREQ)* including its own ID information to obtain a secret key. The secret key is computed by multiple KGCs that received the KREQ and then returned to the requesting node by *key reply packets (KREP)*. A node communicates with only a single KGC, but the secret key is generated via interaction between multiple KGCs. In the secure route discovery phase, each node constructs a connection to a destination node by *secure route request packets (SRREQ)* and *secure route reply packets (SRREP)*. In the secure route maintenance phase, the node finds a disconnection to the destination by *secure route error packets (SRERR)*. In both phases of the secure route discovery and the secure route maintenance, packets include signatures for routing information between a source and its destination. In ISDR+, each node that receives these packets from neighbor nodes generates a signature on the received routing information and then adds the generated signature to the packet. We describe each phase in detail below.

Distributed Key Generation. A node in ISDSR+ needs to obtain a secret key to configure routes to a destination. A node first broadcasts KREQ to KGCs, and then any KGC that receives KREQ responds to the node by returning a global parameter and a master public key for a digital signature scheme.

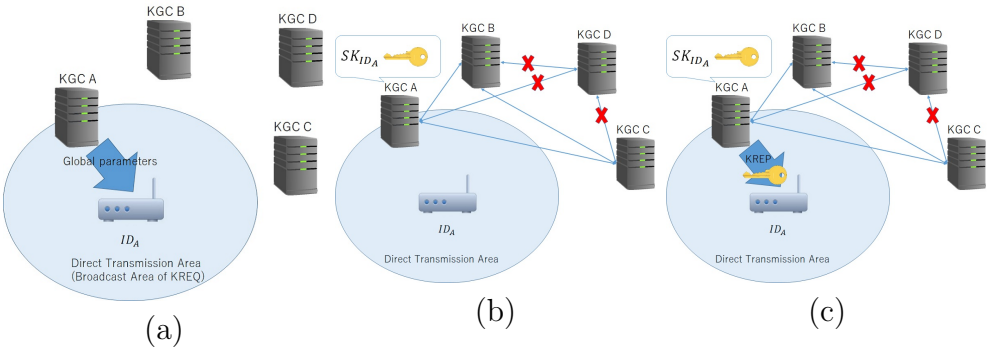


Figure 1: Distributed key generation (initial phase)

The node communicates with only the first KGC that replies, and this KGC interacts with other KGCs to generate a secret key for the node. Even if several KGCs are unavailable, the remaining KGCs can still generate a key. Finally, the KGC returns a secret key as KREP. From the point of view of a node that broadcasts KREQ, it can obtain a secret key as long as any one of the KGCs is within its direct transmission area.

Figure 1 shows the procedure for generating a secret key. Figure 1(a) shows that the node ID_A starts to request key generation to KGCs, and then KGC A responds with the KREQ. Figure 1(b) shows that KGCs start interacting with one another. Even though KGC D is unavailable, KGC A can generate a secret key with the remaining KGCs, i.e., KGC B and KGC C, and can return the generated secret key as KREP in Figure 1(c).

Secure Route Discovery. Each node that receives SRREQ adds its own identity in the received routing information and generates a signature for a signature algorithm. Then, the node broadcasts SRREQ including the signature. These steps are iterated until a destination node receives packets. The destination node then verifies a signature on SRREQ via a verification algorithm for the underlying signature scheme. If the verification returns True, the destination node generates SRREP by generating a signature on the routing information of SRREQ via the signature algorithm. This SRREP is forwarded to the source node of SRREQ along with the routing information. When the source node receives SRREP, the node verifies the received signature via the verification algorithm. If the verification returns True, the source node registers the given route in its memory.

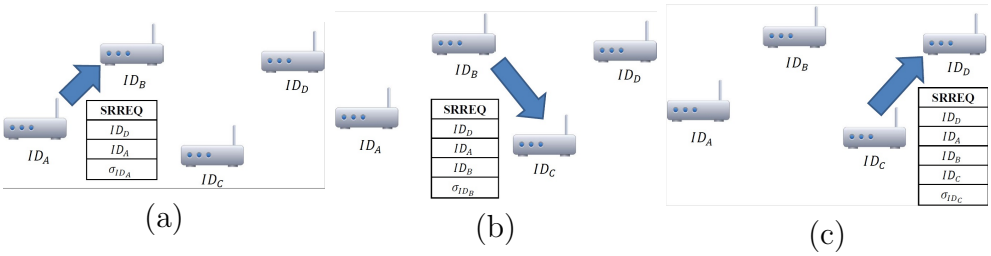


Figure 2: Secure route discovery (search phase)

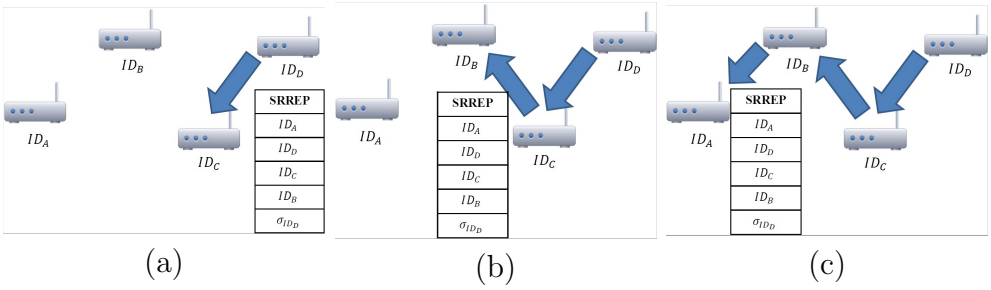


Figure 3: Secure route discovery (reply phase)

Figures 2 and 3 show the procedures for establishing a route. Figures 2(a)–2(c) show that node ID_A starts to search for a route to the destination node ID_D. In Figure 2(a), the node ID_A generates a signature σ_{ID_A} via **Signing**, attaches σ_{ID_A} to an SRREQ, and then broadcasts the SRREQ. In Figure 2(b), the node ID_B receives an SRREQ and generates a signature σ_{ID_B} containing σ_{ID_A}. In Figure 2(c), the node ID_C does the same process as the node ID_B. After the node ID_D receives a SRREQ from ID_C, the node ID_D verifies the received SRREQ. Figure 3(a) shows that the node ID_D sends a SRREP to reply to the node ID_A. The signature σ_{ID_D} is generated only by the node ID_D. The node ID_C forwards a received packet, as in Figure 3(b). In Figure 3(c), the node ID_A receives a packet SRREP and verifies σ_{ID_D} via **Verification**.

Secure Route Maintenance. When any node (we call such node ID for convenience) finds a disconnection to a neighbor on a route registered in its source node, ID sends *secure route error packets (SRERR)* including a signature generated via **Signing** to the source. The intermediate nodes between ID

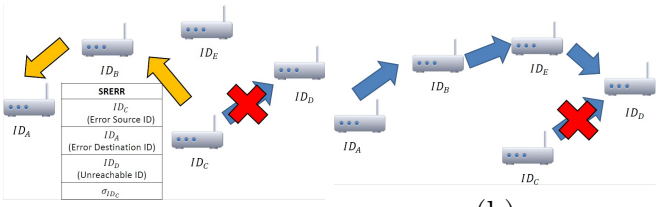


Figure 4: Secure route maintenance

and the source only forward SRERR when they receive the packets. The source node verifies the signature via **Verification** taking the ID as a public key and removes the route if the verification returns True. Figures 4(a) and 4(b) represent the procedures for secure route maintenance. When the node ID_C detects a disconnection, the node ID_C generates an SREER. The signature σ_{ID_C} is generated by only ID_C via **Signing** in Figure 4. After the node ID_A receives an SRERR, it verifies σ_{ID_C} via **Verification**. If the verification passes, ID_A starts to search for other routes to the node ID_D.

3.2. Requirements and Assumptions

We first discuss assumptions used in the proposed model. Networks include several key generation centers (KGCs). Cryptographic parameters utilized in a digital signature scheme are shared to all KGCs in advance. Each device communicates with other devices wirelessly, and this communication is not encrypted except during an initial phase for generating secret keys.

We now define system requirements for ISDSR+ as follows:

- Unforgeability: An attacker cannot maliciously generate SRREQ, SREP, and SRERR (We describe the details in the next paragraph).
- Compactness: The size of signatures for SRREQ, SRREP, and SRERR is fixed with respect to the number of hops. Moreover, device-dependent extra information should not be given in the packets.
- Completeness: All signatures (or their related information) have to be given in SRREQ, SRREP, and SRERR to guarantee the validity of routing information for any source and any destination.
- Availability: Any fixed infrastructure should not be used. Moreover, a key generation function should be available even if multiple KGCs

become unavailable; unless the number of available KGCs is fewer than a threshold designated in advance.

Unforgeability is the main requirement for secure routing protocols. ISDSR (Muranaka et al., 2016) satisfied *Compactness*, *Completeness*, and Unforgeability. In this work, we additionally introduce *Availability* as dynamic configuration of networks for ad hoc networks.

Attack Model. We define an attack model for ISDSR+. In this model, we try to prevent a forgery of routing information with signatures against the malicious forwarding of packets. To do this, we focus on two standpoints, i.e., SRREQ from a source is unforgeable and both SRREP and SRERR to a destination are unforgeable. We assume that a source is always honest since the main motivation of a secure routing protocol is to deliver packets from a source to a destination.

Our target adversary is an *active attacker* (Hu et al., 2005). First, we assume the existence of secure channels between honest nodes and KGCs[2]. Then, the attacker can eavesdrop, inject, or modify all SRREQ, SRREP, and SRERR and their signatures for any target node adaptively. Moreover, the attacker can setup new nodes in networks, and it can compromise several KGCs less than a threshold designated in advance[3]. The attacker then owns all secret keys of the compromised nodes, and it can share the keys among these nodes. Finally, we assume that the attacker is not capable of forging an ID-based sequential aggregate signature scheme with distributed key generation (IBSAS-DKG) described in the next section.

4. Identity-Based Sequential Aggregate Signatures with Distributed Key Generation

In this section, we propose an ID-based sequential aggregate signature scheme with distributed key generation (IBSAS-DKG) scheme as a new building block of a secure routing protocol. This scheme does not require a centralized KGC but can keep a cryptographically compact chain, as presented in a previous ID-based sequential aggregate signature scheme (Boldyreva et al.,

[2]If an attacker can arbitrarily eavesdrop on communication for key generation, the security can be broken easily.

[3]If an attacker can compromise all KGCs without restrictions, then the protocol can be trivially broken.

Algorithm 1 Setup

Ensure: Public Parameter *para*

1: Generate a pairing parameter $(p, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$.
2: $P \leftarrow \mathbb{G}$
3: Choose hash functions $\mathbf{H}_1, \mathbf{H}_2 : 0,1^* \rightarrow \mathbb{G}$ and $\mathbf{H}_3 : 0,1^* \rightarrow \mathbb{Z}_p^*$
4: Set $para = (p, \mathbb{G}, \mathbb{G}_T, \mathbf{e}, P, t, \{\mathbf{H}_i\}_{i=1,\ldots,3})$

2010). We describe only the construction in Algorithms 1–5, and define syntax of IBSAS-DKG and its security in Appendix A. Hereinafter, we assume that each node has a unique index.

In this scheme, we utilize bilinear maps and bilinear groups defined as follows. Let $\mathbb{G}$ and $\mathbb{G}_T$ be groups with a common prime order p. A bilinear map $\boldsymbol{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a map such that the following conditions hold: For all $U, V \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p^*$, $\boldsymbol{e}(aU, bV) = \boldsymbol{e}(U, V)^{ab}$; For any generator $P \in \mathbb{G}$, $\boldsymbol{e}(P, P) \neq 1_{\mathbb{G}_T}$, $1_{\mathbb{G}_T}$ is an identity element over $\mathbb{G}_T$; There is an efficient algorithm that can compute $\boldsymbol{e}(U, V)$ for any $U, V \in \mathbb{G}$. In this paper, we assume that a discrete logarithm problem (DLP) in $\mathbb{G}$ and $\mathbb{G}_T$ is hard, and say that $\mathbb{G}$ is a bilinear group if all the conditions described above hold. We call the parameter $(p, \mathbb{G}, \mathbb{G}_T, \boldsymbol{e})$ a *pairing parameter*.

We also utilize appropriate coefficients by the Lagrange interpolation, i.e., the Lagrange coefficients, and denote the coefficients for some i by $\lambda_j = \prod_{j \in \mathcal{D}} \frac{-i}{j-i}$ for any set $\mathcal{D}$.

We show the correctness, i.e., signatures can be verified correctly, in Appendix A.3. In addition, the security of the proposed IBSAS-DKG scheme can be proven via formal proofs, which we show in Appendix A.4.

5. Implementation

In this section, we explain the implementation of two signature algorithms, i.e., Algorithms 1–5 and the RSA signature algorithm. In our previous paper (Kojima and Yanai, 2017), we implemented the signing algorithm of ISDSR using the JPBC library, in which elliptic curves and pairing functions are written in Java. In this work, we use a PBC library[4] implemented in C language with a Java wrapper provided by JPBC to improve the calculation time. We then conduct a refactoring of our previous implementation.

[4]https://crypto.stanford.edu/pbc/

Algorithm 2 TKeyGen

Require: Public Parameter $para$, Threshold t, Number n of Nodes, Set $\mathcal{D} = [1, n]$ of Nodes, Index $i \in \mathcal{D}$.

Ensure: Master Public Key mpk and Partial Secret Key x_i for i

1: $(a_{1,i,j}, a_{2,i,j}) \leftarrow \mathbb{Z}_p$ for all $j \in [0, t]$
2: Set a polynomial function $\mathbf{F}_i(x) = \sum_{j=0}^{t} a_{1,i,j} x^j$, where $\mathbf{F}_i(0) = a_{1,i,0}$
3: Set a polynomial function $\mathbf{G}_i(x) = \sum_{j=0}^{t} a_{2,i,j} x^j$, where $\mathbf{G}_i(0) = a_{2,i,0}$
4: Node i sends $\mathbf{F}_i(j)$ and $\mathbf{G}_i(j)$ to all $j \in \mathcal{D} \backslash \{_i\}$
5: Node i receives $\mathbf{F}_i(j)$ and $\mathbf{G}_i(j)$ from all $j \in \mathcal{D} \backslash \{i\}$
6: $a_{1,i} = \sum_{j \in \mathcal{D}} \mathbf{F}_j(i)$, $a_{2,i} = \sum_{j \in \mathcal{D}} \mathbf{G}_j(j, i)$
7: Node i sends $(a_{1,i,0}P, a_{2,i,0}P)$ to all $j \in \mathcal{D} \backslash \{i\}$
8: Node i receives $(a_{1,j,0}P, a_{2,j,0}P)$ from all $j \in \mathcal{D} \backslash \{i\}$
9: Set $x_i = (a_{1,i}, a_{2,i})$
10: $(a_1 P, a_2 P) = \left(\sum_{j \in \mathcal{D}} a_{1,j,0} P, \sum_{j \in \mathcal{D}} a_{2,j,0} P \right) \in \mathbb{G}^2$
11: $mpk = (a_1 P, a_2 P)$

Algorithm 3 UTKeyGen

Require: Public Parameter $para$, Master Public Key mpk, Identity $ID \in \{0, 1\}^*$, Set $\Omega \subseteq \mathcal{D}$ of Nodes such that $|\Omega| \geq t$.

Ensure: Secret Key sk_{ID} for ID

1: Send ID to all $j \in \Omega$
2: For all $j \in \Omega$, $a_{1,j}\mathbf{H}_1(ID)$, $a_{2,j}\mathbf{H}_2(ID)$
3: Receive $(a_{1,j}\mathbf{H}_1(ID), a_{2,j}\mathbf{H}_2(ID)) \in \mathbb{G}^2$ from all $j \in \Omega$
4: $a_1\mathbf{H}_1(ID) = \left(\sum_{j \in \Omega} \lambda_j a_{1,j} \mathbf{H}_1(ID) \right) \in \mathbb{G}$
5: $a_2\mathbf{H}_1(ID) = \left(\sum_{j \in \Omega} \lambda_j a_{2,j} \mathbf{H}_2(ID) \right) \in \mathbb{G}$
6: $sk_{ID} = (a_1\mathbf{H}_1(ID), a_2\mathbf{H}_1(ID))$

ISDSR+ and RSA-based signature algorithms are implemented in Java as the three applications below. $APP_{ISDSR+-J}$ is an implementation of the signature algorithm described in Section 4 with JPBC. Next, $APP_{ISDSR+-P}$ is also implemented with JPBC, and uses a Java wrapper API for the PBC native library provided by JPBC. The wrapper plays the role of an interface between JPBC and PBC, and it enables $\text{APP}_{ISDSR+-P}$ to execute the calculation of elliptic curves and pairing functions with the PBC library. Finally, APP_{RSA} is an implementation for the RSA-based signature algorithm. This

Algorithm 4 Signing

Require: Public Parameter $para$, Secret Key sk_{ID_i}, Message $m \in \{0,1\}^*$, Signature σ'

Ensure: Signature σ'

1: Parse σ as $(\sigma_1, \sigma_2, \sigma_3)$
2: If $i = 1$, set $\sigma = (0, 0, 0)$
3: $(r, x) \leftarrow \mathbb{Z}_p^2$
4: $\sigma'_3 = \sigma_3 + x \cdot P$
5: $\sigma'_2 = \sigma_2 + r \cdot P$
6: $\sigma'_1 \leftarrow \sigma_1 + r\sigma_3 + x\sigma'_2 + a_2\mathbf{H}_2(ID_i) + \mathbf{H}_3(ID_i||m_i)a_1\mathbf{H}_1(ID_i)$

Algorithm 5 Verification

Require: Public Parameter $para$, Master Public Key mpk, List $((ID_1, m_1), \ldots, (ID_N, m_N))$ of Identities and Messages, Signature σ

Ensure: True or False

1: Parse σ as $(\sigma_1, \sigma_2, \sigma_3)$
2: Check if all $ID_1, \ldots, ID_N$ are distinct
3: Check $\mathbf{e}(\sigma_1, P) = \mathbf{e}(\sigma_2, \sigma_3) \cdot \mathbf{e}\left(\sum_{i=1}^{N} \mathbf{H}_2(ID_i), a_2P\right) \cdot \mathbf{e}\left(\sum_{i=1}^{N} \mathbf{H}_3(ID_i||m_i)\mathbf{H}_1(ID_i), a_1P\right)$
4: **return** If both checks are true, True
5: **return** If not, False

application is implemented in Java and its signing and verification functions are executed with the RSA algorithm API of the Java security package.

These three applications have four functions., i.e., signature generation, signature verification, packet management, and sending/receiving packets. The packet management function transforms an instance of a packet class to a byte array to send the packet class data as datagram packets or vice versa. Each application equips an instance of `DatagramSocket` class to send and to receive packets. Figure 5 shows a class diagram of the three applications, where the eight classes are parts of the whole classes of the applications. We explain the `Node` and `AbstractAlgorithm` classes below.

Classes for a node. `Node` class is designed for one node that equips a communication socket and a signature algorithm and that has members and methods for sending data, receiving data, signing, and verification. A member `rcv`,

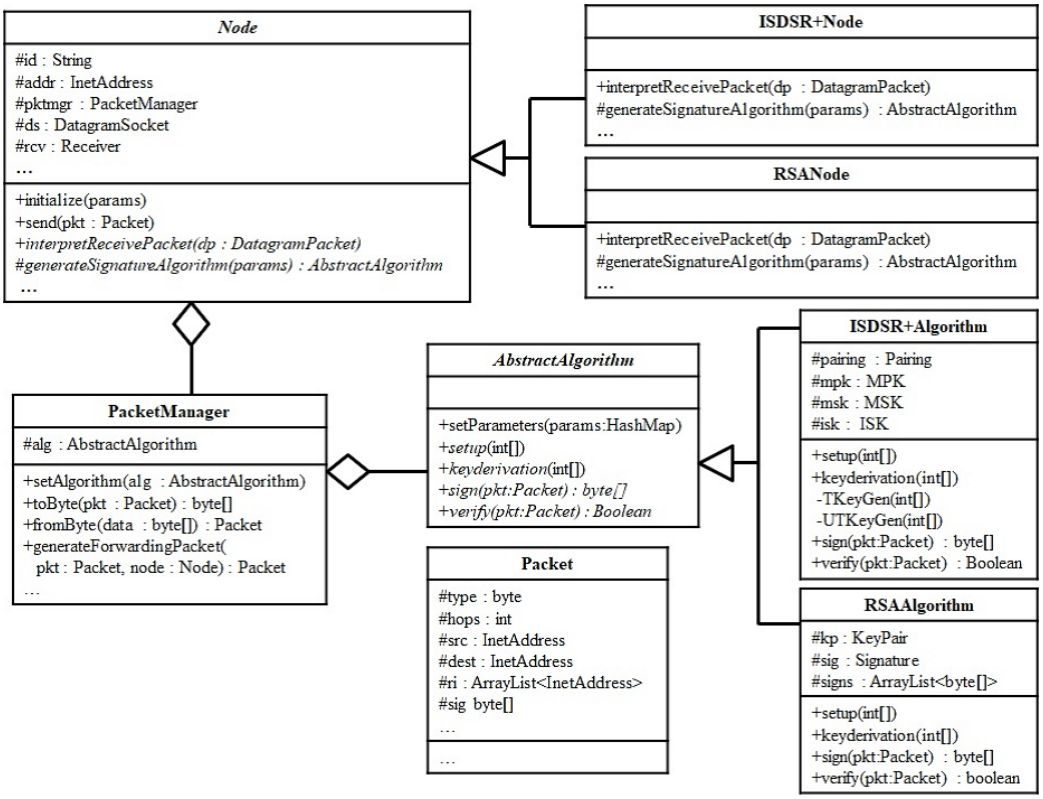


Figure 5: A class diagram of the applications

which is an instance of the `Receiver` class, implements a `Runnable` interface to run as a receiver thread with a member `ds`. An abstract method `interpretReceivedPacket(dp)` is called when `rcv` receives a packet as a `DatagramPacket` from `ds`. The received packet is interpreted through the method `interpretReceivePacket(dp)` implemented in the `ISDSRNode` class and the `RSANode` class. The constructor method calls `generateSignatureAlgorithm(param)` to assign its own signature algorithm to a member `pkitmgr` through a method `PacketManager.setAlgorithm(alg)`. Each method `generateSignatureAlgorithm(param)` of `ISDSR+Node` and of `RSANode` instantiates an instance of the signature algorithm, such as the `ISDSRAlgorithm` and the `RSAAlgorithm`, respectively. After each constructor finishes, the member `pktmgr` has an appropriate instance of the child class of `AbstractAlgorithm`.

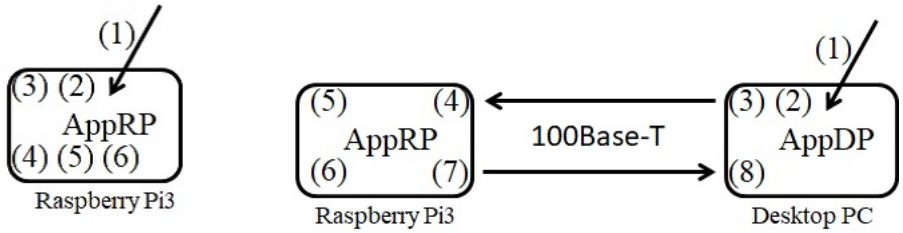
Classes for a signature algorithm. The `AbstractAlgorithm` class is an abstract class for implementing concrete signature algorithm classes, i.e., the `ISDSR+Algorithm` and the `RSAAlgorithm`, and has four abstract methods, namely, `setup(int)`, `keyderivation(int)`, `sign(pkt)`, and `verify(pkt)`. These four methods must be implemented in each concrete class for signature generation and verification using its own signature algorithm.

The `ISDSR+Algorithm` class applies JPBC version 2.0.0 (De Caro and Iovino, 2011) for calculating elliptic curves and pairing functions for signature generation and verification of ISDSR+. JPBC is a port of PBC that performs the mathematical operations underlying pairing-based cryptosystems directly in Java. A constructor method of the `ISDSR+Algorithm` class instantiates a member pairing to use pairing functions and executes Algorithms 1–3 to generate keys, including a master public key, partial secret keys, and a secret key for each user. The `RSAAlgorithm` class uses the RSA algorithm APIs in Java security package, which provides algorithms for signing, verifying, encoding, and decoding. In a constructor method of `RSAAlgorithm`, members `kp` and `sig` are generated from methods in the Java security package.

Classes for a packet and packet management. The `Packet` class has information to find a route from the source node to its destination node. The `PacketManager` class has two methods, `toByte(pkt)` and `fromByte(data)`, to transform a byte array received by a `Node.ds` to a `Packet` class instance or vice versa. This class has methods for generating a request packet and a reply packet. When `Node` class receives a packet, the `Node` class calls the method `pktmgr.generateForwardingPacket` to generate a forwarding packet. The method `generateForwardingPacket(pkt,node)` generates a forwarding packet using a member `alg` to verify signatures in the `pkt` or to generate signatures for a new forwarding packet.

6. Experiments

We now evaluate the calculation time for distributed key generation and round trip times (hereinafter RTTs) based on our implementation described in the previous section. The execution time of distributed key generation, i.e., KREQ and KREP, is measured on a Raspberry Pi, which is a device that can estimate performances of actual applications by allowing transmissions within real wireless communication, such as Bluetooth and Near-field Communication (NFC). Meanwhile, experiments to measure RTTs between nodes, i.e., SRREQ, SRREP, and SRRER, are conducted to run



(a) Distributed Key Generation

(b) Round Trip Times

Figure 6: A Schematic of the procedure for experiments

$\mathrm{APP}_{ISDSR+-J}$, $\mathrm{APP}_{ISDSR+-P}$, and APP_{RSA} on a Raspberry Pi3. The experiments are also conducted in an actual environment including a Raspberry Pi and a desktop PC via LAN connection. In all the experiments described above, we applied "a.properties" in JPBC as parameters to use an elliptic curve and a pairing function in our application program. Here, the bit length of an output group $\mathbb{G}_T$ of a pairing parameter, i.e., a security level, of "a.properties" is 1024 bits. Although such a level is weak in a general usage of a pairing parameter, it is sufficient for secure routing protocols in ad hoc networks. In particular, we can just keep routes secure during a short period when packets are trying to reach their destination.

6.1. Experimental Environment

We conducted experiments with a Raspberry Pi and a desktop PC described below. The Raspberry Pi is a Raspberry Pi3 model B with Ubuntu 14.4 as OS and Java version of OpenJDK 1.8.1_151. The desktop PC is equipped with Xeon(R) E5-2667 v3 3.20GHz as CPU and 512 GB memory, and it has CentOS 7 as OS and Java version of OpenJDK 1.8.1_151. The desktop PC and the Raspberry Pi connect with a switching hub with a 100 Base-T wired LAN.

Figure 6 shows the procedures of an experiment. Hereinafter, we call behaviors of our implemented application program on a desktop PC *AppDP* and those on a Raspberry Pi3 *AppRP*. In Figure 6(a), the digits (1)–(6) indicate AppRP processes. On the other hand, in Figure 6(b), the digits (1), (2), (3), and (8) indicate AppDP processes, and the digits (4)–(7) indicate AppRP processes. We describe the behaviors of each experiment below.

First, the behaviors of KREQ and KREP in Figure 6(a) are described as follows: (1) AppRP is given a value n as the number of KGCs, and then it

generates n instances; (2) AppRP generates polynomial functions for each instance; (3) All instances in AppRP distribute values for the polynomial functions with each other; (4) All instances in AppRP generate a partial secret key; (5) Some instances in AppRP generate a master public key; and (6) Some instances in AppRP are given an ID and then generate a secret key for the ID.

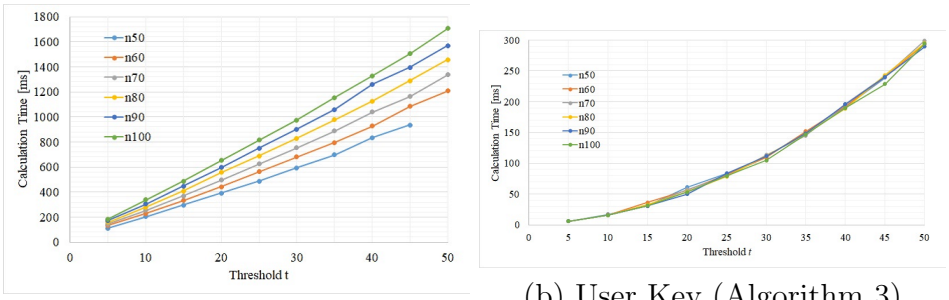
The behaviors of SRREQ and SRREP in Figure 6(b) are described as follows: (1) AppDP is given a value N as the number of nodes; (2) AppDP generates a packet including routing information that contains N; (3) AppDP sends the generated packet; (4) AppRP receives a packet; (5) AppRP verifies signatures in the received packet; (6) After verification passes, AppRP generates its own signatures from the contents of the received packet, and then generates a forwarding packet that includes the signatures and routing information; (7) AppRP sends the generated packet; and (8) AppDP receives a packet and verifies the received packet. The environment described above is common even for SRRER because computational behaviors for each node is almost the same as those of SRREQ and SRREP.

6.2. Result of Distributed Key Generation

In this section, we measure calculation times for distributed key generation. All results of the experiments are averaged over 10 runs of each application. One experiment represents the case where the AppRP executes both a KREQ and a KREP. Processes that are identical to a KREQ and a KREP are represented by (6) in Figure 6(a). The other digits represent the case where the AppRP executes threshold key generation of a partial secret key and a master public key, i.e., Algorithm 2.

Figures 7(a) and 7(b) show the results of threshold key generation for KGCs and of secret key generation for each user. In particular, Figure 7(a) show the results of the experiments where the AppRP executed processes (1)–(5) in Figure 6(a), and Figure 7(b) show those where the AppDP executed process (6) in Figure 6(a). The horizontal axis represents threshold values t, and the lines represent different numbers of KGCs n.

The calculation time of generating a master key is linear in proportion to t and n as shown in Figure 7(a), whereas the calculation time of generating secret key for a user is quadratic with respect to t as shown in Figure 7(b). For example, as shown in Figure 7(a), the calculation time of threshold key generation for KGCs can be estimated at $(0.25 * n + 8) * t$ milliseconds in the current environment. On the other hand, the results in Figure 7(b) are



(a) Master Key (Algorithm 2)

(b) User Key (Algorithm 3)

Figure 7: Computational Time for Distributed Key Generation

independent of n, and the calculation time of secret key generation for a user can be estimated at $(0.09 * t + 1.1) * t$ milliseconds.

Using the estimation described above in a case where $t = 10$ and $n = 100$, the calculation time of threshold key generation for KGCs can be estimated to be 330 milliseconds, which is identical to Figure 7(a). Moreover, the calculation time of secret key generation for a user can be estimated to be 19 milliseconds, which is identical to Figure 7(b).

6.3. Round Trip Time

In this section, we measure round trip times (RTTs) for the route discovery step. All results of the experiments are averaged over 100 runs of each application. One experiment represents the case where the AppRP forwards an SRREQ, which is generated by the received SRREQ with no verification. Processes that forward an SRREQ are represented by (4), (6), and (7) in Figure 6(b). Here, the AppRP does not verify a received packet before the AppRP generates a forwarding SRREQ. The other experiment represents the case where the AppRP replies an SRREP, which is generated by the received SRREQ with verification. Processes that reply the SRREP are represented by (4)–(7) on the AppRP in Figure 6(b). This experiment indicates that the AppRP is the destination node of the received packet. The AppRP needs to verify the received packet to guarantee routing information in the received packet.

Figures 8 and 9 show the results when we applied the parameter file "a.properties" to both the $\mathrm{APP}_{ISDSR+-J}$ and the $\mathrm{APP}_{ISDSR+-P}$. Meanwhile, the APP_{RSA} needs the key length as a parameter, and we use 1024 bits, which provides the same security level as that of ISDSR+, for the results in

Figure 10. We also note that the results are also useful for SRRER because its behaviors are almost the same as described above.

Case: Forwarding SRREQ. Figures 8(a), 9(a), and 10(a) show results of the experiments where the AppRP executed processes (4), (6), and (7) in Figure 6. The received SRREQ is not verified before generating a forwarded SRREQ. The horizontal axis represents the number of nodes, i.e., the number of nodes in packets sent by the AppDP.

Figure 8(a) shows the results of the $\mathrm{APP}_{ISDSR+-J}$, which uses JPBC to the generation and verification of signatures. Figure 9(a) shows the results of $\mathrm{APP}_{ISDSR+-P}$, which was computed in the PBC library in C language. The RTTs in Figures 8(a) and 9(a) are approximately 0.24 seconds and 0.1 seconds, respectively. These results show that the RTTs of ISDSR+ are not dependent on the number of nodes and that the RTTs of the $\mathrm{APP}_{ISDSR+-P}$ are faster than those of the $\mathrm{APP}_{ISDSR+-J}$. Figure 10(a) shows the results of the APP_{RSA}, in which the key length is 1024 bits. Compared with Figure 9(a), the RTTs in Figure 10(a) increase when the number of nodes increases. However, the RTTs in Figure 9(a) are similar at approximately 0.1 seconds even when the number of nodes increases. Based on these results, we can see that ISDSR+ with the PBC library in C language is superior to the RSA-based signature algorithm.

Case: Replying SRREP. In this case, we assume that the AppRP is the destination node of a received SRREQ. The AppRP has to verify the received SRREQ to guarantee routing information stored in the SRREQ. Then, the AppRP sends the generated SRREP to reply to the source node in the SRREQ. Figures 8(b), 9(b), and 10(b) show the RTTs when the AppRP executes processes (4)–(7) in Fig. 6. In Figure 8(b), we conducted the experiment until 10 nodes because the calculation time of $\mathrm{APP}_{ISDSR+-J}$ is significantly large. In particular, the RTT increases by approximately 0.8 seconds every additional node. On the other hand, although The RTT in Figure 10(b) increases with the number of nodes, the largest RTT is approximately 0.3 seconds.

6.4. Further Potential Performance

For distributed key generation, a case with a small threshold t and a large number n of whole KGCs is better in a general sense of the availability. The calculation time of threshold key generation for KGCs, i.e., Algorithm 2,

can be normally separated from configuration of routes from a source to a destination. The calculation time of secret key generation for each user, i.e., Algorithm 3, is independent of the number n of KGCs as shown in Section 6.2. We can hence adopt an arbitrary pair of a large n and a small t. For instance, in a case where $t = 5$ and $n = 10000$, a master key for a KGC and a secret key for a user can be computed within approximately 12.4 seconds and 8 milliseconds, respectively. If we use, for example, NFC with its maximum transmission speed of 424 kbit/s or Bluetooth with 24 Mbit/s in the connection between each device and KGCs and 100BASE-T LAN in the connection between the KGCs, transmission of a secret key from the KGCs to a device would take less than 15 milliseconds.

The experiments in this paper were conducted on a highly reliable network environment with low latency and low packet drop rate. In this environment, the packets were received completely even when the length of data in a datagram packet was very large. In a practical case where the density of nodes is high, collisions occur easily in the datalink layer. In the datalink layer, the number of frames is large when a packet size is large. A packet sent by the APP_{RSA} needs more frames than that of the APP_{ISDSR+} in the datalink layer because of its packet size. If the frame drop rates of the APP_{RSA} and the APP_{ISDSR+} are the same, then the packet drop rate of the APP_{ISDSR+} is lower than that of the APP_{RSA}. Although the APP_{RSA} is superior to the APP_{ISDSR} based on their RTTs, the APP_{RSA} does not work well in unreliable networks because of its large packet size.

We also performed refactoring on our previous implementation to introduce preprocessors for elliptic curve calculations in Algorithm 4. Algorithm 4 has six steps, and the steps after step 4 require elliptic curve calculations, e.g., $x \cdot P$ and $r \cdot P$. The variable P and a_1 are preprocessed using the API of the JPBC named `jpbc.element.getElementPowPreProcessing()` at the same time when the `ISDSRAlgorithm` class is instantiated. The refactoring makes the signing process of ISDSR+ faster.

7. Discussion

In this section, we briefly discuss how ISDSR+ can achieve requirements described in Section 3.2 and show potential applications of ISDSR+.

Achievement of Requirements. The four requirements described in Section 3.2 rely on features of IBSAS-DKG. In particular, given its signature compression

and the use of IDs for public keys, the compactness can be achieved as well as the completeness by the guarantee of routing information from any source to any destination. Moreover, by proving the security of IBSAS-DKG and combining it with the Kim-Tsudik framework (Kim and Tsudik, 2009) described in Section 2, the unforgeability of ISDSR+ can be achieved. Finally, the availability can be achieved by introducing distributed key generation with arbitrary (t, n) setting. Specifically, adopting a large n and a small t essentially enables any device to join the protocol anytime and anywhere. Moreover, any user can receive a secret key by connecting to any KGC within its direct transmission area. We thus conclude that our proposed ISDSR+ can achieve the requirements.

Application of ISDSR+. We focus on the RTTs of the applications. By applying the PBC library to the $\mathrm{APP}_{ISDSR+-P}$, the RTTs of the $\mathrm{APP}_{ISDSR+-P}$ become shorter than those of the APP_{RSA} as long as there are no verification cases. We consider that the signature verification should be executed on a specific destination, e.g., the sink node in sensor networks because it has computational resources that outperform other nodes. For example, the railway monitoring system (Velagandula et al., 2017) is a bridge health monitoring system that involves collection of data from wireless sensors installed on a bridge to a remote server for damage identification in the bridge. In this system, a remote server typically has high computational power and can therefore verify packets sent by other sensors.

8. Conclusion

In this paper, we proposed a new protocol ISDSR+ by introducing distributed key generation in ISDSR and implementing APP_{ISDSR+} and APP_{RSA} on a Raspberry Pi3. Compared with ISDSR, ISDSR+ does not require a centralized key generation center and therefore achieves infra-less setting.

Our main building blocks include a novel signature scheme named IBSAS-DKG, which supports the features of ISDSR+ as described in Section 3. We also conducted experiments to measure RTTs. The experiments were classified into without verification and with verification after receiving a packet. In the case of without verification, all results of the $\mathrm{APP}_{ISDSR+-P}$ were superior to that of the APP_{RSA}.

Finally, as presented in section 7, our experimental environment is more reliable in terms of frame loss rate compared with an environment in practice.

Therefore, as future work, we will conduct experiments in an environment that is not highly reliable by using mininet (Lantz et al., 2010) or mininet-wifi (Fontes et al., 2015). In that case, we will measure the packet delivery ratio between a source node and a destination node.

Acknowledgment

This research was supported in part by the Japan Society for the Promotion of Science KAKENHI Number 16K16065.

References

Ács, G., 2009. Secure Routing in Multi-Hop Wireless Networks. Ph.D. thesis. Budapest University of Technology and Economics.

Bellare, M., Neven, G., 2006. Multi-signatures in the plain public-key model and a general forking lemma, in: Proc. of CCS 2006, ACM. pp. 390–399.

Boldyreva, A., 2003. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme, in: Proc. of PKC 2003, Springer. pp. 31–46.

Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.H., 2010. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing (extended abstract). URL: `http://www.cc.gatech.edu/~aboldyre/papers/bgoy.pdf`.

Boneh, D., Gentry, C., Lynn, B., Shacham, H., 2003. Aggregate and verifiably encrypted signatures from bilinear maps, in: Proc. of EUROCRYPT 2003, Springer. pp. 416–432.

Boneh, D., Lynn, B., Shacham, H., 2001. Short signatures from the weil pairing, in: Proc. of ASIACRYPT 2001, Springer. pp. 514–532.

Buttyán, L., Vajda, I., 2004. Towards provable security for ad hoc routing protocols, in: Proc. of SASN, ACM Press. pp. 94–105.

De Caro, A., Iovino, V., 2011. jpbc: Java pairing based cryptography, in: Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011, IEEE, Kerkyra, Corfu, Greece, June 28 - July 1. pp. 850–855.

Desmedt, Y., Frankel, Y., 1989. Threshold cryptosystems, in: Proc. of CRYPTO 1989, Springer-Verlag. pp. 307–315.

Fontes, R.R., S.Afzal, Brito, S.H.B., Santos, M.A.S., Rothenberg, C.E., 2015. Mininet-wifi: Emulating software-defined wireless networks, in: Proc. 11th International Conference on Network and Service Management (CNSM), pp. 384–389.

Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T., 1996. Robust threshold dss signatures, in: Proc. of EUROCRYPT 1996, Springer-Verlag. pp. 354–371.

Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T., 1999. Secure distributed key generation for discrete-log based cryptosystems, in: Proc. of EUROCRYPT '99, Springer-Verlag. pp. 295–310.

Gentry, C., Ramzan, Z., 2006. Identity-based aggregate signatures, in: Proc. of PKC 2006, Springer. pp. 257–273.

Ghosh, U., Datta, R., 2011. Identity based secure aodv and tcp for mobile ad hoc networks, in: Proc. of ACWR 2011, ACM. pp. 339–346.

Ghosh, U., Datta, R., 2013. Sdrp: Secure and dynamic routing protocol for mobile ad-hoc networks. IET Network 3, 235–243.

Giorgetti, A., Lucchi, M., Tavelli, E., Barla, M., Gigli, G., Casagli, N., Chiani, M., Dardari, D., 2016. A robust wireless sensor network for landslide risk analysis: System design, deployment, and field testing. IEEE Sensors Journal 16, 6374–6386. doi:10.1109/JSEN.2016.2579263.

Hu, Y.C., Perrig, A., Johnson, D., 2002. Ariadne: a secure on demand routing protocol for ad hoc network, in: Proc. of MobiCom 2002, ACM.

Hu, Y.C., Perrig, A., Johnson, D., 2005. Ariadne: a secure on demand routing protocol for ad hoc network. Wireless Networks 11, 21–38.

Itakura, K., Nakamura, K., 1983. A public-key cryptosystem suitable for digital multi-signatures. NEC Research and Development 71, 1–8.

Johnson, D., Hu, Y., Maltz, D., 2007a. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. IETF RFC4728.

Johnson, D., Hu, Y.C., Maltz, D., 2007b. The dynamic source routing protocol (dsr) for mobile ad hoc networks for ipv4.

Karlof, C., Wagner, D., 2003. Secure routing in wireless sensor networks: Attacks and countermeasures. Ad Hoc Networks 1, 293–315.

Kato, N., 2014. Advanced ad hoc and mesh networks technologies to facilitate specialized disaster-resilient networks, in: Proc. of Trustcom 2014. Keynote.

Kim, J., Tsudik, G., 2009. Srdp: Secure route discovery for dynamic source routing in manets. Ad Hoc Networks 7, 1097–1109.

Kojima, H., Yanai, N., 2017. Performance evaluation for the signature algorithm of isdsr on raspberry pi, in: Proc. 10th International Workshop on Autonomous Self-Organizing Networks,conjunction with CANDAR'17.

Lantz, B., Heller, B., Mckeown, N., 2010. A network in a laptop: Rapid prototyping for software-defined networks, in: In ACM SIGCOMM HotNets Workshop.

Lee, Y.H., Kim, H., Chung, B., Lee, J., Yoon, H., 2003. On-demand secure routing protocol for ad hoc network using id based cryptosystem, in: Proc. of 4th ICPDCAT, IEEE. pp. 211–215.

Li, M., Liu, Y., 2008. Underground coal mine monitoring with wireless sensor networks. ACM Transactions on Sensor Networks 5.

Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H., 2004. Sequential aggregate signatures from trapdoor permutations, in: Proc. of EUROCRYPT 2004, Springer. pp. 74–90.

May, B., 2009. Ariadne: Secure on-demand routing in ad-hocnetworks - an explanation for dummies. `http://www.net.t-labs.tu-berlin.de/teaching/ss09/IR_seminar/talks/ariadne_may.handout.pdf`.

Muranaka, K., Yanai, N., Okamura, S., Fujiwara, T., 2015. Secure routing protocols for sensor networks: Construction with signature schemes for multiple signers, in: Proc. of Trustcom 2015, IEEE. pp. 1329–1336.

Muranaka, K., Yanai, N., Okamura, S., Fujiwara, T., 2016. ISDSR: Secure DSR with ID-based Sequential Aggregate Signature, in: Proc.of ICETE 2016, pp. 376–387. doi:`10.5220/0006001003760387`.

Papadimitratos, P., Haas, Z.J., 2002. Secure routing for mobile ad hoc networks, in: Proc. of CNDS 2002, pp. 27–31.

Park, C., Kurosawa, K., 1996. New elgamal type threshold digital signature scheme. IEICE Transactions on Fundamentals E79-A, 86–93.

Perkins, C., Belding-Royer, E., Das, S., 2003a. Ad hoc On-demand Distance Vector (AODV) Routing. IETF RFC3561.

Perkins, C., Belding-Royer, E., Das, S., 2003b. Ad hoc on-demand distance vector (aodv) routing. URL: `\url{https://www.ietf.org/rfc/rfc3561.txt}`.

Sanzgiri, K., LaFlamme, D., Dahill, B., Levine, B.N., Shields, C., Belding-Royer, E., 2005. Authenticated routing for ad hoc networks. IEEE Journal on Selected Areas in Communications 23, 598–610.

Shamir, A., 1984. Identity-based cryptosystems and signature schemes, in: Proc. of CRYPTO 84, Springer. pp. 47–53.

Song, J., Kim, H., Lee, S., Yoon, H., 2005. Security enhancement in ad hoc network with id-based cryptosystem, in: Proc. of ICACT 2005, IEEE. pp. 372–376.

Velagandula, S.D., Dhang, N., Datta, R., Ghosh, S.K., Maroju, S., 2017. Railway bridge health monitoring system using smart wireless sensor network, in: Proc. of WiSec 2017, ACM. pp. 288–290.

Yanai, N., Manbo, M., Okamoto, E., 2013. Ordered multisignature schemes under the cdh assumption without random oracles, in: Proc. of ISC 2013, Springer. pp. 367–377.

Zapata, M., Asokan, N., 2002. Securing ad hoc routing protocols, in: Proc. of WISE, ACM Press. pp. 1–10.

Zhou, L., Haas, Z., 1999. Securing ad hoc network. IEEE Network Magazine 13, 24–30.

Appendix A. Formal Discussion of IBSAS-DKG

Appendix A.1. Syntax

In this section, we define the syntax of a IBSAS-DKG scheme and its correctness.

Definitions of Algorithms. The algorithms of an IBSAS-DKG scheme are defined as follows. Here, let a message space and an identity space be $\mathcal{M}$ and $\mathcal{ID}$, respectively. We assume that each KGC owns a unique index $i \in \mathbb{N}$, where $\mathbb{N}$ is a set of integers, and a set of KGCs is then defined as a subset of $\mathbb{N}$. We also denote by $\mathcal{D} \subseteq \mathbb{N}$ any subset of KGCs which own partial secret keys corresponding to a master public key.

Setup: Given a security parameter 1^κ, output a public parameter $para$.

TKeyGen: Given security parameters $(t, n) \in \mathbb{N}^2$, $para$, a set $\mathcal{D} \subseteq \mathbb{N}$ of n KGCs, and an index $i \in \mathcal{D}$, via interaction with all $j \in \mathcal{D} \backslash \{i\}$, output a pair (x_i, mpk) of a partial secret key msk as a private output for i and a master public key as a common output for $\mathcal{D}$.

UTKeyGen: Given $para, mpk, ID$ and a set $\Omega \subseteq \mathcal{D}$, output a secret key sk_{ID} corresponding to ID.

Sign: Given $para$, a secret key sk_{ID_i}, ID_i, a message $m_i \in \mathcal{M}$ to be signed, a set $L = \{(m_j, ID_j)\}_{j=1}^{i-1}$ of pairs of signed messages and their identities, and an aggregate signature σ, return a new aggregate signature σ' on a new set $L' = L \cup \{(m_i, ID_i)\}$ or $\perp$ to indicate an error.

Verify: Given $para, mpk$, a set $L = \{(m_j, ID_j\}_{j=1}^{i}$ of pairs of signed messages and their identities, and an aggregate signature σ, output True or False.

Correctness. The correctness of the IBSAS-DKG scheme described above is defined as follows.

Definition 1 (Correctness)**.** For all $para \leftarrow \mathbf{Setup}(1^\kappa)$, all $(t, n) \in \mathbb{N}^2$, all $ID_i \in \mathcal{ID}$, all $m_i \in \mathcal{M}$, all $\Omega \subseteq \mathcal{D} \subseteq \mathbb{N}$ such that $|\mathcal{D}| = n$ and $|\Omega| \geq t$ hold, all $L \subseteq \mathcal{M} \times \mathcal{ID}$, and all $(x_j, mpk) \leftarrow \mathbf{TKeyGen}(t, n, para, \mathcal{D}, j)$ where $j \in \mathcal{D}$ holds, the following condition hold:

$$\text{True} = \mathbf{Verify}\left(para, mpk, L', \mathbf{Sign}\left(\begin{array}{c} para, \\ \mathbf{UTKeyGen}\begin{pmatrix} para, mpk, \\ ID_i, \Omega \end{pmatrix}, \\ ID_i, m_i, L, \sigma \end{array}\right)\right),$$

where $L' = L \cup \{(m_i, ID_i)\}$. We say that an IBSAS-DKG scheme is correct if the condition described above holds.

Appendix A.2. Security Definitions

In this section, we define security of an IBSAS-DKG scheme. In particular, we define unforgeability of signatures and robustness of distributed key generation as properties of an IBSAS-DKG scheme. The former property is necessary for preventing a signature forgery via received signatures. On the other hand, the second property can prevent an adversary corrupting several KGCs less than a threshold from obtaining the capability of key generation. These two properties are important for the underlying purpose of an IBSAS-DKG scheme, i.e., guarantee of the validity of routing information and key generation without a fixed infrastructure.

Appendix A.2.1. Unforgeability

We define the unforgeability of an IBSAS-DKG scheme via the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. Namely, an advantage of $\mathcal{A}$ can be obtained with a probability that $\mathcal{C}$ outputs *accept* in the game. Hereinafter, we denote by $x^{(i)}$ a value of the i-th query for all x.

Initial Phase: The challenger $\mathcal{C}$ generates a public parameter $para \leftarrow \mathbf{Setup}(1^\kappa)$, and chooses integers $(t, n) \in \mathbb{N}^2$ and a set $\mathcal{D} \subseteq \mathbb{N}^n$ of n KGCs. Next, $\mathcal{C}$ generates $(x_i, mpk) \leftarrow \mathbf{TKeyGen}(t, n, para, \mathcal{D}, i)$ for all $i \in \mathcal{D}$. $\mathcal{C}$ then runs $\mathcal{A}$ with $(t, n, para, mpk, \mathcal{D})$ as input.

Corrupt Query: $\mathcal{A}$ sends any index $i^{(h)} \in \mathcal{D}$ to $\mathcal{C}$, and $\mathcal{C}$ returns a partial secret key $x_{i^{(h)}}$ for the given index $i^{(h)}$.

KeyDer Query: $\mathcal{A}$ sends any string $\left(ID^{(h)}, \Omega \subseteq \mathcal{D}\right)$ to $\mathcal{C}$, and $\mathcal{C}$ returns a secret key $sk_{ID^{(h)}}$ for $ID^{(h)}$.

Sign Query: $\mathcal{A}$ generates a signing query $\left(para, mpk, m^{(h)}, ID^{(h)}, L, \sigma\right)$. Given the query, $\mathcal{C}$ returns a signature σ.

Output After q_c iterations of the **Corrupt Query**, q_k iterations of the **KeyDer Query**, and q_s iterations of the **Sign Query**, $\mathcal{A}$ outputs a forgery (L^*, σ^*), where $L = \{(ID_i^*, m_i^*)\}_{i=1}^N$ where $N \in \mathbb{N}$ and the following conditions hold: the **Verify** algorithm outputs True; $q_c < t$ holds; there is exactly one $ID_{i^*}^*$ such that $ID_{i^*}^* \notin \{ID_i^{(h)}\}_{h=1}^{q_k}$ holds for the **KeyDer Query**; for the $ID_{i^*}^*$, a tuple of $(m_{i^*}^*, ID_{i^*}^*, L_{i^*-1}^* = \{(ID_j^*, m_j^*)\}_{j=1}^{i^*-1})$ has never been queried to the **Sign Query**; and all $ID^* \in L^*$ are distinct. If all the conditions hold, then $\mathcal{C}$ outputs *accept*. Otherwise, $\mathcal{C}$ outputs $reject$.

Definition 2. We say that an IBSAS-DKG scheme is $(t, n, q_c, q_k, q_s, q_h, N, \epsilon)$-unforgeable if there is no probabilistic polynomial-time adversary $\mathcal{A}$ who forges with $(t, n, q_c, q_k, q_s, q_h, N, \epsilon)$. Here, we say that $\mathcal{A}$ forges the scheme with $(t, n, q_c, q_k, q_s, q_h, N, \epsilon)$ if a challenger $\mathcal{C}$ outputs *accept*, in the security game described above, with a probability greater than ϵ on a threshold t and a number n of KGCs. Here, $\mathcal{A}$ can generate at most q_c corruption queries, at most q_k key derivation queries, at most q_s signing queries, and at most q_h random oracle queries, and N is the number of signers in the $\mathcal{A}$'s output and queries.

Appendix A.2.2. Robustness

We define the robustness via the following simulation between a real scheme and an ideal scheme for an adversary $\mathcal{A}$. More specifically, a challenger $\mathcal{C}$ executes either the algorithms defined in Section Appendix A.1 or algorithms without threshold computations. Then, the robustness is defined as follows:

Definition 3. We say that an IBSAS-DKG scheme is robust if for all *mpk* there are **KeyGen** and **UKeyGen** whose distributions are identical for any probabilistic polynomial-time algorithm $\mathcal{A}$ to **TKeyGen** and **UTKeyGen**, respectively. Here, $\mathcal{A}$ can corrupt up to $t-1$ signers in order to obtain $t-1$ partial secret keys.

Appendix A.3. Correctness of the Scheme

In this section, we briefly show that our scheme in Section 4 is correct in the meaning of Appendix A.1. In particular, we first check if a secret key

generated by the **UTKeyGen** algorithm is identical to a part of the **Verify** algorithm, and then show that the **Verify** algorithm outputs True via the resultant secret key.

Theorem 1. *The proposed scheme is correct.*

Proof. First, for any ID and its related secret key, the following equations hold via the Lagrange interpolation:

$$\begin{aligned}
\mathbf{e}\left(\sum_{j\in\Omega}(\lambda_j\alpha_{1,j}H_1(ID)),P\right) &= \mathbf{e}\left(a_1H_1(ID)),P\right) = \mathbf{e}\left(H_1(ID)),a_1P\right), \\
\mathbf{e}\left(\sum_{j\in\Omega}(\lambda_j\alpha_{2,j}H_2(ID)),P\right) &= \mathbf{e}\left(a_2H_2(ID)),P\right) = \mathbf{e}\left(H_2(ID)),a_2P\right).
\end{aligned}$$

For the **Verify** algorithm, the verification equation can be written as follows: for any $i, n(\geq i) \in \mathbb{Z}^2$:

$$\begin{aligned}
\mathbf{e}(\sigma_1,P) &\stackrel{?}{=} \mathbf{e}\left(rxP+\sum_{i=1}^{n}a_2\mathbf{H}_2(ID_i)+\sum_{i=1}^{n}\mathbf{H}_3(ID_i||m_i)a_1\mathbf{H}_1(ID_i),P\right) \\
&= \mathbf{e}(rxP,P)\cdot\mathbf{e}\left(\sum_{i=1}^{n}a_2\mathbf{H}_2(ID_i),P\right)\cdot\mathbf{e}\left(\sum_{i=1}^{n}\mathbf{H}_3(ID_i||m_i)a_1\mathbf{H}_1(ID_i),P\right) \\
&= \mathbf{e}\left(xP,rP\right)\cdot\mathbf{e}\left(\sum_{i=1}^{n}\mathbf{H}_2(ID_i),a_2P\right)\cdot\mathbf{e}\left(\sum_{i=1}^{n}\mathbf{H}_3(ID_i||m)\mathbf{H}_1(ID_i),a_1P\right)
\end{aligned}$$

The computation step described above is identical to the verification equation on the **Verify** algorithm. The algorithm then returns True. The proposed scheme is thus correct because the two conditions described in Section Appendix A.1. □

Appendix A.4. Security Analysis of IBSAS-DKG

We analyze the security of the proposed scheme by formal proofs. We first prove that the scheme is unforgeable under the security assumptions named an ID-based sequential aggregate signature computational Diffie-Hellman (IBSAS-CDH) assumption Boldyreva et al. (2010) defined below. Next, We prove that the scheme is robust in the sense of the definitions in Section Appendix A.2.

Definition 4 ((q,ϵ)-IBSAS-CDH Assumption in $\mathbb{G}$)**.** We define an IBSAS-CDH problem with a security parameter 1^k as follows: for a pairing parameter $(p,\mathbb{G},\mathbb{G}_T,\boldsymbol{e})$ and a given tuple (P,a_1P,a_2P,b_1P,b_2P) with uniformly random $(a_1,a_2,b_1,\ b_2)\leftarrow\mathbb{Z}_p^4$ as input compute $(rxP+a_1b_1P+ma_2b_2P,\ rP,xP)$ for uniformly random $(r,x)\leftarrow\mathbb{Z}_p$ under accessing to an oracle $\mathcal{O}^{\mathrm{IBSAS-CDH}}_{P,a_1P,a_2P,b_1P,b_2P}$ that takes $m\in\mathbb{Z}_p$ as input and returns $(rxP+a_1b_1P+ma_2b_2P,rP,xP)$ for randomly generated numbers $(r,x)\leftarrow\mathbb{Z}_p^2$, where an element m involved in each query must be different from the element m involved in the final output. We say that a (q,ϵ)-IBSAS-CDH assumption in $\mathbb{G}$ holds if there is no probabilistic polynomial-time algorithm that can solve the IBSAS-CDH problem with a probability greater than ϵ. Here, the algorithm can generate at most q queries to the oracle.

Theorem 2. *Suppose that* $\mathbf{H}_i$ *for* $i\in[1,3]$ *is modeled as a random oracle. The proposed scheme is* $(t,n,q_c,q_j,q_k,q_s,q_r,q_{h_1},q_{h_2},q_{h_3},N,\epsilon)$*-unforgeable under the* (q,ϵ')*-IBSAS-CDH assumption in* $\mathbb{G}$*, where* $q\le q_s$*,*

$$\epsilon' \ \ge\ \epsilon\left(1-\frac{q_c+1}{n}\right)\frac{1}{e(N(q_s+1)+q_k+1)},$$

e is the base of the natural logarithm.

Proof. Given $(p,\mathbb{G},\mathbb{G}_T,\boldsymbol{e},P,a_1P,a_2P,b_1P,b_2P)$ and access to an oracle $\mathcal{O}^{\mathrm{IBSAS-CDH}}_{P,a_1P,a_2P,b_1P,b_2P}$, $\mathcal{B}$ sets $(p,\mathbb{G},\mathbb{G}_T,\boldsymbol{e},P)$ as *para* and (a_1P,a_2P) as *mpk*. Next, $\mathcal{B}$ sets $\mathcal{ID}[-,-,-]$, $\mathcal{D}[-,-,-,-,-]$, $\mathbf{H}_1[-,-,-],\mathbf{H}_2[-,-,-]$ and $\mathbf{H}_3[-,-]$ as an $\mathcal{ID}$-list, $\mathcal{D}$-list, a $\mathbf{H}_1$-list, a $\mathbf{H}_2$-list, and a $\mathbf{H}_3$-list, respectively. Then, $\mathcal{B}$ guesses some index $i^*\in[1,n]$. For $i\in[1,n]\backslash\{i^*\}$, $\mathcal{B}$ chooses polynomial functions $\mathbf{F}_i(x)$ and $\mathbf{G}_i(x)$ with the degree t, and computes $a_{1,i},a_{2,i},a_{1,i,0}P$ and $a_{2,i,0}P$. $\mathcal{B}$ then computes $a_{1,i^*,0}P=a_1P-\sum_{i\in[1,n]\backslash\{i^*\}}a_{1,i,0}$ and $a_{2,i^*,0}P=a_2P-\sum_{i\in[1,n]\backslash\{i^*\}}a_{2,i,0}$ as a partial secret key x_i. Next, $\mathcal{B}$ registers $(ID_i,\mathbf{F}_i(x),\mathbf{G}_i(x),a_{1,i},a_{2,i})$ in the $\mathcal{D}$-list for $i\in[1,n]\backslash\{i^*\}$, and registers $(ID_{i^*},-,-,-,-)$ in the $\mathcal{D}$-list for i^*. $\mathcal{B}$ sets $\mathcal{D}=[1,n]$, and runs $\mathcal{A}$ with $(p,\mathbb{G},\mathbb{G}_T,\boldsymbol{e},P,a_1P,a_2P,b_1P,b_2P,\mathcal{D})$ in the following manner. Here, $\mathcal{B}$ utilize a random coin $B\in\{0,1\}$ with a probability ε to set 1, and we finally determine ε to complete the proof:

$\mathbf{H}_1$ Query $(ID_i^{(h)})$: This oracle simulation is associated with the **$\mathbf{H}_2$ Query**. If $(ID_i^{(h)})$ has been registered in the $\mathbf{H}_2$ list, $\mathcal{B}$ retrieves B_i from the list. Otherwise, $\mathcal{B}$ sets $B_i=1$ with the probability ε or $B_i=0$ with the probability $1-\varepsilon$. Next, $\mathcal{B}$ generates $\alpha_i\leftarrow\mathbb{Z}_p^*$, and then sets $\mathbf{H}_1(ID^{(h)})=\alpha_iP$

for $B_i = 1$ or $\mathbf{H}_1(ID^{(h)}) = \alpha_i P + b_1 P$ for $B_i = 0$. $\mathcal{B}$ registers $(ID_i^{(h)}, \alpha_i, B_i)$ in the $\mathcal{H}_1$-list, and returns the $\mathbf{H}_1(ID^{(h)})$.

$\mathbf{H}_2$ Query $(ID_i^{(h)})$**:** This oracle simulation is associated with the $\mathbf{H}_1$ **Query**. If $(ID_i^{(h)})$ has been registered in the $\mathbf{H}_1$ list, $\mathcal{B}$ retrieves B_i from the list. Otherwise, $\mathcal{B}$ sets $B_i = 1$ with the probability ε or $B_i = 0$ with the probability $1-\varepsilon$. Next, $\mathcal{B}$ generates $\beta_i \leftarrow \mathbb{Z}_p^*$, and then sets $\mathbf{H}_2(ID^{(h)}) = \beta_i P$ for $B_i = 1$ or $\mathbf{H}_2(ID^{(h)}) = \beta_i P + b_2 P$ for $B_i = 0$. $\mathcal{B}$ registers $(ID_i^{(h)}, \beta_i, B_i)$ in the $\mathcal{H}_2$-list, and returns the $\mathbf{H}_2(ID^{(h)})$.

$\mathbf{H}_3$ Query $(ID_i^{(h)} \parallel m_i^{(h)})$**:** $\mathcal{B}$ generates $\delta_i \leftarrow \mathbb{Z}_p^*$ and sets $\mathbf{H}_3(ID_i^{(h)} \parallel m_i^{(h)}) = \delta_i$. $\mathcal{B}$ then registers $(ID_i^{(h)} \parallel m_i^{(h)}, \delta_i)$ in the $\mathcal{H}_3$-list, and returns the $\mathbf{H}_3(ID_i^{(h)} \parallel m_i^{(h)})$.

Corrupt Query $(i^{(h)})$**:** $\mathcal{B}$ first checks if $i^{(h)} = i^*$. If so, $\mathcal{B}$ aborts the process. Otherwise, $\mathcal{B}$ retrieves a partial secret key $x_i = (a_{1,i}, a_{2,i})$ from the $\mathcal{D}$-list, and returns x_i.

KeyDer Query $\left(ID^{(h)}, \Omega \subseteq \mathcal{D}\right)$**:** $\mathcal{B}$ checks if $ID^{(h)}$ has been registered in the $\mathbf{H}_1$-list and the $\mathbf{H}_2$-list. If not, $\mathcal{B}$ executes $\mathbf{H}_1$ **Query** and $\mathbf{H}_2$ **Query**. Next, $\mathcal{B}$ checks if $B_i = 0$ on the lists for $ID^{(h)}$. If so, $\mathcal{B}$ aborts the process. If not, i.e., $B_1 = 1$, $\mathcal{B}$ computes $sk_{ID^{(h)}} = (\alpha_i a_1 P, \beta_i a_2 P)$. This $sk_{ID^{(h)}}$ can be written as $a_1 \mathbf{H}_1(ID^{(h)}) = \alpha_i a_1 P$ and $a_2 \mathbf{H}_2(ID^{(h)}) = \beta_i a_2 P$ from the $\mathbf{H}_1$-list and the $\mathbf{H}_2$-list, respectively. Finally, $\mathcal{B}$ registers $\left(ID^{(h)}, a_1 \mathbf{H}_1(ID^{(h)}), a_2 \mathbf{H}_2(ID^{(h)})\right)$ in the $\mathcal{ID}$-list, and returns a secret key $sk_{ID^{(h)}}$.

Sign Query $\left(para, mpk, m_i^{(h)}, ID_i^{(h)}, L, \sigma\right)$**:** $\mathcal{B}$ checks if $B_i = 1$ for $ID_i^{(h)}$. If so, $\mathcal{B}$ executes **KeyDer Query** and then generates a signature in the same manner as that of the original **Sign** algorithm. This distribution is exactly identical to that of the original algorithm except for the use of random oracles because $\mathcal{B}$ can know a secret key.

Otherwise, i.e., $B_i = 1$ for $ID_i^{(h)}$, $\mathcal{B}$ checks if L contains ID_j for $j \in [1, i-1]$ such that $B_j = 0$ holds on the $\mathbf{H}_1$-list and the $\mathbf{H}_2$-list. $\mathcal{B}$ aborts the process if so. Otherwise, $\mathcal{B}$ discards the given σ, and retrieves δ_i for $ID_i^{(h)} \parallel m_i^{(h)}$ from the $\mathbf{H}_3$-list. $\mathcal{B}$ then receives $(rxP + a_1 b_1 P + \delta_i a_2 b_2 P, rP, xP)$ by accessing to the oracle $\mathcal{O}^{\text{IBSAS-CDH}}_{P, a_1 P, a_2 P, b_1 P, b_2 P}$ with δ_i as input. $\mathcal{B}$ computes $rxP + a_1 b_1 P + \delta_i a_2 b_2 P + \sum_{j=1}^{i} (\alpha_j a_1 P + \delta_j \beta_j a_2 P)$ as σ_1 by retrieving α_j and β_j for all $j \in [1, i]$ from the $\mathbf{H}_1$-list and the $\mathbf{H}_2$-list. $\mathcal{B}$ also sets $\sigma_2 = rP$ and $\sigma_3 = xP$. This σ_1 can be written as $rxP + \sum_{j=1}^{i} (a_1 \mathbf{H}_1(ID_j) + a_2 H_3(ID_j \parallel m_j) \mathbf{H}_2(ID_j))$ from simulation of each list. The signature $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ is accepted on the

verification algorithm, and thus its distribution in the simulation described above is indistinguishable for $\mathcal{A}$. $\mathcal{B}$ returns a signature $\sigma = (\sigma_1, \sigma_2, \sigma_3)$.

Output: After the simulation described above, $\mathcal{A}$ outputs a forgery (L^*, σ^*) where there exists exactly a single $ID^*_{j^*}$, whose secret key has never been queried to **KeyDer Query**, from the definition of the forgery. $\mathcal{B}$ checks if, for $ID^*_{j^*}$, $B_{j^*} = 0$ holds on the $\mathbf{H}_1$-list and the $\mathbf{H}_2$-list and, for other $ID^*_j \in L^*$, $B_{j^*} = 1$ holds. If the statement is false, $\mathcal{B}$ aborts the process. Otherwise, the signature can be written as follows because the verification holds:

$$\sigma_1 = rxP + a_1 b_1 P + \delta_{j^*} a_2 b_2 P + \sum_{i=1}^{n} (a_1 \alpha_i P + \delta_i a_2 \beta_i P).$$

$\mathcal{B}$ can hence extract a solution to the problem by following computation:

$$\sigma' = \sigma_1 - \sum_{i=1}^{n} (\alpha_1 a_1 P + \delta_i \beta_i a_2 P) = rxP + a_1 b_1 P + \delta_{j^*} a_2 b_2 P,$$

where δ_{j^*} is a value which has never been queried to the oracle. The tuple of $(\sigma', \sigma_2, \sigma_3)$ is a solution to the IBSAS-CDH problem.

To complete the proof, we analyze a success probability ϵ' of $\mathcal{B}$. In addition to a success probability ϵ of $\mathcal{A}$, there are five cases where $\mathcal{B}$ aborts the process; the first case $abort_C$ is in the **Corrupt Query** where a partial secret key for i^* has been queried; the second case $abort_K$ is in the **KeyDer Query** where a secret key with $B_i = 0$ in the $\mathbf{H}_1$-list or the $\mathbf{H_2}$-list has been queried; the third case $abort_S$ is in the **Sign Query** where there are multiple identities with $B_i = 0$ in the $\mathbf{H}_1$-list or the $\mathbf{H_2}$-list for any i; and, the firth case $abort_{out}$ is in the **Output** where the statement about B_{j^*} is false. The success probability can be then estimated as follows:

$$\begin{aligned}
\epsilon' &= \epsilon \cdot \Pr[\bigwedge_{j=1}^{q_c} \neg abort_C] \cdot \Pr[\bigwedge_{j=1}^{q_k} \neg abort_K] \cdot \Pr[\neg abort_S] \cdot \Pr[\neg abort_{out}] \\
&\geq \epsilon \cdot \left(\frac{n-1}{n} \cdot \frac{(n-1)-1}{n-1} \cdots \frac{(n-q_c)-1}{(n-q_c)} \right) \cdot (\varepsilon^{q_k}) \cdot \left(\varepsilon^{Nq_s}\right) \cdot \left(\varepsilon^{(N-1)}(1-\varepsilon)\right) \\
&\geq \epsilon \cdot \left(\frac{(n-q_c)-1}{n} \right) \cdot \varepsilon^{q_k + Nq_s + N - 1}(1-\varepsilon) \\
&\geq \epsilon \cdot \left(1 - \frac{q_c + 1}{n} \right) \cdot \varepsilon^{q_k + N(q_s+1)}(1-\varepsilon).
\end{aligned}$$

The variable ε is finally determined in order to optimize the probability described above. Here, let $f(\varepsilon)$ be a function $\varepsilon^z(1-\varepsilon)$ where $z = q_k + N(q_s + 1)$. Then, $f(\varepsilon)$ is maximized at $\varepsilon_{opt} := \frac{z}{z+1}$ according to the derived function. That is, the following inequation can be obtained for the function $f(\varepsilon)$.

$$f(\varepsilon_{opt}) = \left(\frac{z}{z+1}\right)^a \left(1 - \frac{z}{z+1}\right) = \left(1 + \frac{1}{z}\right)^{-z} \left(\frac{1}{z+1}\right) \geq e^{-1} \left(\frac{1}{z+1}\right),$$

where e is the base of the natural logarithm. The success probability ϵ' is then bounded as follows:

$$\epsilon' \geq \epsilon \cdot \left(1 - \frac{q_c + 1}{n}\right) \cdot \left(\frac{1}{e(q_k + N(q_s + 1) + 1)}\right).$$

The probability is polynomially bounded.

□

Theorem 3. *The proposed scheme is robust.*

Proof. In this proof, we show existence of two algorithms, **KeyGen** and **UKeyGen**, whose distributions are indistinguishable from **TKeyGen** and **UTKeyGen**, which have threshold setting. The algorithms are constructed as follows:

KeyGen: Given $para$, generate $(a_1, a_2) \leftarrow \mathbb{Z}_p^2$ as msk and compute $(a_1P, a_2P) \in \mathbb{G}^2$ as mpk.

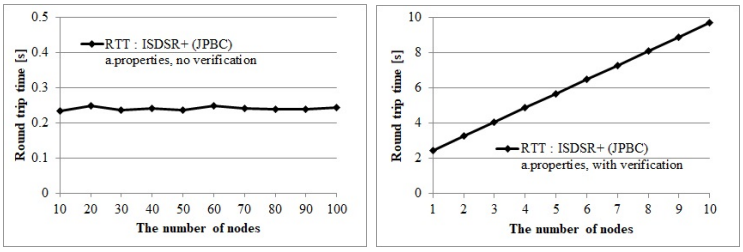
UKeyGen: Given $(para, mpk, ID, msk)$, generate $sk_{ID} = (a_1\mathbf{H}_1(ID_i), a_2\mathbf{H}_2(ID_i))$.

For **KeyGen**, $mpk = (a_1P, a_2P)$ is uniformly distributed as long as $msk = (a_1, a_2)$ is uniform-randomly generated. On the other hand, for **TKeyGen**, $mpk = (\sum_{j\in\mathcal{D}} a_{1,j,0}P, \sum_{j\in\mathcal{D}} a_{2,j,0}P) \in \mathbb{G}^2$ and $msk = (a_1 = \sum_{ID_i\in\mathcal{D}} a_{1,i,0}, a_2 = \sum_{ID_i\in\mathcal{D}} a_{2,i,0}) \in \mathbb{Z}_p^2$) are uniform-randomly generated if there exists at least a single node which honestly generates a partial secret key. Thus, these distributions are indistinguishable.

Next, for **UKeyGen**, if at least t partial secret keys are collected, the output $sk_{ID} = (a_1\mathbf{H}_1(ID_i), a_2\mathbf{H}_2(ID_i))$ computed with the Lagrange interpolation is indistinguishable from that of **UTKeyGen** because their outputs are exactly identical. Furthermore, if there exists at least a single node which honestly generates a partial secret key, mpk and msk are uniform-randomly generated. Thus, these distributions are indistinguishable.

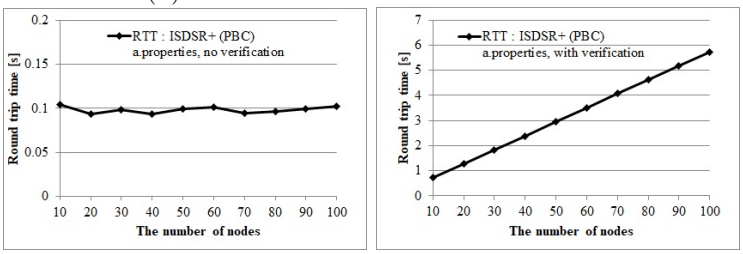
Finally, we briefly note that the algorithms described above is uncomputable for an adversary. In particular, computations for **KeyGen** and

UKeyGen corresponds to a discrete logarithm problem and CDH problems. Although we omit the details, the algebraic structures are almost identical to that of the BLS signatures (Boneh et al., 2001) which are provably secure. □



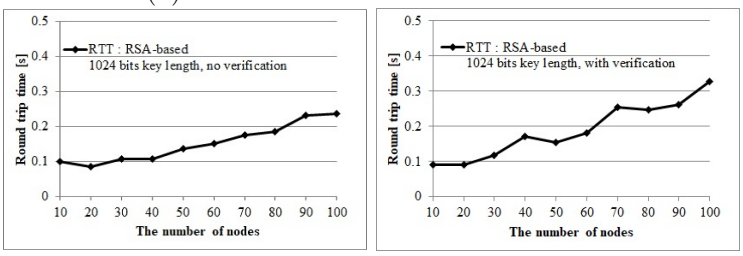
(a) (b)

Figure 8: RTT of $\text{APP}_{ISDSR+-J}$ (JPBC) in ISDSR+ against the number of nodes (a) without verification and (b) with verification



(a) (b)

Figure 9: RTT of $\text{APP}_{ISDSR+-P}$ (PBC) in ISDSR+ against the number of nodes (a) without verification and (b) with verification



(a) (b)

Figure 10: RTT of RSA-based algorithm against the number of nodes (a) without verification and (b) with verification