sensors

MDPI

Article

Anonymous Asynchronous Ratchet Tree Protocol for Group Messaging †

Kaiming Chen , Jiageng Chen * and Jixin Zhang

School of Computer, Central China Normal University, NO. 152 Luoyu Road, Wuhan 430079, China; cvk907@gmail.com (K.C.); zhangjixxx@foxmail.com (J.Z.)

* Correspondence: chinkako@gmail.com

† This paper is an extended version of the conference paper: Chen, K.; Chen, J. Anonymous End to End Encryption Group Messaging Protocol Based on Asynchronous Ratchet Tree. In Proceedings of the 22nd International Conference on Information and Communications Security (ICICS 2020), Copenhagen, Denmark, 24–26 August 2020; pp. 588–605.

Abstract: Signal is the first application that applies the double ratchet for its end-to-end encryption protocol. The core of the double ratchet protocol is then applied in WhatsApp, the most popular messaging application around the world. Asynchronous Ratchet Tree (ART) is extended from ratchet and Diffie-Hellman tree. It is the first group protocol that applies Forward Secrecy (FS) with Post-Compromised Security (PCS). However, it does not consider protecting the privacy of user identity. Therefore, it makes sense to provide anonymous features in the conditions of FS and PCS. In this paper, the concepts of Internal Group Anonymity (IGA) and External Group Anonymity (EGA) are formalized. On the basis of IGA and EGA, we develop the "Anonymous Asynchronous Ratchet Tree (AART)" to realize anonymity while preserving FS and PCS. Then, we prove that our AART meets the requirements of IGA and EGA as well as FS and PCS. Finally, the performance and related issues of AART are discussed.

Keywords: end-to-end encryption; forward secrecy; post-compromised security; anonymity; group messaging protocol

check for updates

Citation: Chen, K.; Chen, J.; Zhang, J. Anonymous Asynchronous Ratchet Tree Protocol for Group Messaging. *Sensors* **2021**, *21*, 1058. https://doi.org/10.3390/s21041058

Academic Editor: James (Jong Hyuk) Park

Received: 28 November 2020
Accepted: 26 January 2021
Published: 4 February 2021

1. Introduction

1.1. Background

With the help of Internet development, Instant Messaging (IM) applications are much important in people's lives. According to statistics, WhatsApp is the most popular IM application around the world with more than 2 billion active users. Facebook Messenger has 1.3 billion users. The third is WeChat with about 1 billion. In 2018, people spent 27.6 h a week online, of which 15.6% was used for instant messaging. In addition, WeChat is the second IM application of China, and LINE is popular in East Asian countries. A large amount of data containing personal privacy information will be generated through these platforms.

End-to end encryption (E2EE) is used to protect user privacy such that the server or any attackers cannot read messages during the communication of IM. When the secret key is not compromised, Indistinguishability under Chosen Ciphertext Attack (IND-CCA) is considered as a standard to protect IM communication, in which case an attacker can request a prepared ciphertext [1]. However, when the secret key is compromised, there should be Forward Secrecy (FS) [2] and Post-Compromised Security (PCS) [3]. FS is to ensure that the adversary cannot obtain the key or plaintext information of the past secret messages. PCS is to guarantee that after multiple interactions, the compromised communication will be restored to a secure state again.

The group message protocol is extended from one-to-one IM with at least three users during the communication. The sender transmits a message, and the other group members will receive the corresponding one. Many protocols of IM applications directly send the message ciphertext, the encryption key, and the ciphertext of the key to each member with the one-to-one secure protocol. This strategy is called “sender keys”. Because the session key is determined by the sender, all members should keep the connection with others. This operation cannot meet the requirements of PCS because the receiver should obtain the identity of the sender to apply “sender keys”. To deal with this issue, ART protocol [4] is designed, which is based on the ideas of point-to-point [5] and stateful [3] protocols.

However, there are still issues. In 2019, WhatsApp was hacked through its phone call bug, which led to user information being leaked [6]. Thus, the user’s identity may be disclosed because of the engineering loopholes in the implementation of applications, and anonymous features are required. Current group message protocol cannot provide FS, PCS, as well as anonymity at the same time. Therefore, we aim to propose a protocol that can satisfy FS, PCS, and anonymity. According to the conference version [7], we re-formalize the two anonymous features, External Group Anonymity (EGA) and Internal Group Anonymity (IGA), as attack games to resist internal and external group attackers. In EGA, communications among different groups cannot be distinguished. Therefore, in EGA, attackers who are not members of a group cannot link users to the appropriate group. When the key is leaked, the external attacker can be regarded as a member of the group. EGA cannot resist such attackers. Therefore, IGA is required, in which other members cannot accurately locate the message sender except for the messages sent by themselves.

1.2. Contributions

In this paper, we develop the structure of ART to satisfy IGA security and apply the one-time address [8] to achieve the security of EGA. We formalize our construction with the algorithms $Init$ to create the group channel, Enc to encrypt and send messages, and Dec to receive and decrypt messages. The sub-algorithm SKG is to derive the session key, and $Update$ and $UpdateGpk$ are to update group tree by sender and receiver, respectively. The tools are used to construct the following algorithms: a cipher $\mathcal{E} = (E_{CPA}, D_{CPA})$ which satisfies Indistinguishability under Chosen Plaintext Attack (IND-CPA), a MAC system $I = (S, V)$ to protect the integrity of the message, and (*Send*,*Get*) to send and get messages from the server according to the one-time address. Then, we prove the security of AART that satisfies FS, PCS, and anonymity. Finally, we show that the performance of AART is better than the “sender keys” and pair-wise Signal group protocols and it is close to ART while providing anonymity features.

2. Related Works

In this section, we analyze the group protocols of IM applications and show that these protocols do not provide anonymity along with FS and PCS.

2.1. Group Protocols

2.1.1. iMessage

Apple’s iMessage is the first popular E2EE application, but it turns out to be insecure under IND-CCA [9]. According to iMessage white paper [10], before sender A transmits a message to receiver B via iMessage, A should get the address of B from Apple’s server called APN because APN will store all users’ addresses. Furthermore, The group messaging protocol of iMessage is “sender keys”. Thus, anonymous features cannot be satisfied with iMessage.

2.1.2. LINE

LINE [11] is an E2EE application that is popular in East Asia. According to the protocol of LINE called *Letter Sealing*, there are some issues such as impersonating attacks [12]. In group messaging, a group master key is calculated by the creator and sent to other members via “sender keys”. This master key will not be changed so that if it is compromised, the contents of communication will be revealed by the attacker. Thus, PCS is not satisfied in LINE.

2.1.3. Signal

OTR [13] is the first application to provide ratchet. In ratchet protocol, users negotiate new Diffie-Hellman (DH) keys of each session, and the old session keys will be deleted and cannot be derived again. Signal's protocol is called double ratchet. It proves that double ratchet can satisfy FS and PCS [5]. It can be observed from this protocol that long-term public keys are included in the associated data. So, the identities of users will be disclosed to the message server. Signal's group messaging protocol is pair-wise, which requires that each member should maintain a one-to-one Signal protocol with other members rather than sending keys. Because of the pair-wise protocol, anonymity cannot be satisfied.

2.1.4. ART

ART [4] is extended from ratchet and Diffie-Hellman tree, which first applies FS and PCS to group messaging protocol. The creator of a group generates DH key pairs for others. DH key pairs are set as the leaves of the DH tree, and the parents' DH key pairs are generated from the ones of their children. The public DH tree is sent to other members. When sending messages, the sender needs to refresh his leaf DH key and the public DH keys from the corresponding leaf to the root of the group tree. The new public keys are sent to others to update their DH trees according to the location of the sender. Because the position of the sender is public and bound to the identity of the sender, ART cannot satisfy anonymity.

2.1.5. WeChat and QQ

WeChat [14] and QQ [15] are the most popular IM applications in China. They apply Secure Sockets Layer (SSL) or Transport Layer Security (TLS) to protect the message security. A TLS connection will be set with the server to which the user is logged in. The group messages are transferred through this channel. TLS 1.3 proves to be FS. Though for PCS, because the later session key is derived from the former one, it cannot be satisfied in TLS, the same as WeChat and QQ. It is claimed that the identities of users can be protected. However, they do not offer technical details as well as the source code. Moreover, it is also not clear whether they are E2EE protocols or not.

2.2. Some Anonymous Approaches Applied in E2EE

Tor [16] is an anonymous network composed of many user volunteers. Tok [17] is the IM application based on Tor. When communicating, the sender randomly selects the same volunteer points, then derives long-time session keys of them. These keys are used to encrypt sending messages in sequence. According to the sequence, these messages are passed to the next point and decrypted by each point using the derived key until it is delivered to the receiver. Thus, the address of the sender is only known to the first point. This address of the receiver is only known to the last point. However, FS and PCS cannot be satisfied when the long-term keys are compromised.

Identity-based encryption (IBE) is used to validate and authenticate the anonymous public keys in E2EE [18]. Because of the low efficiency, KEM/DEM is applied to encrypt the secret key of the authenticator [19]. The encrypted secret key is sent to a proxy, then the proxy delays this message to the service provider for validation. As the proxy is trusted, the identity of the sender can be protected. Just like Tor, the secret key of the sender is long-term. So, it cannot provide FS and PCS.

3. Security Definitions

There are fundamental tools for the security definition. $\mathcal{M}$ is the message space. $\mathcal{K}$ is the key space. $\mathcal{C}$ is the cipher space. Σ is the MAC space. $\mathcal{U}$ is finite user identity set. $\mathcal{E} = (E, D)$ is the encryption scheme, $E(k, m) = c : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C}$ is the encryption algorithm, and $D(k, c) = m : \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{M}$ is the decryption algorithm. $I = (S, V)$ is a MAC system where $S(k, c) = \sigma : \mathcal{K} \times \mathcal{C} \rightarrow \Sigma$ and $V(k, (c, \sigma)) = \{0, 1\} : \mathcal{K} \times (\mathcal{C} \times \Sigma) \rightarrow \{0, 1\}$. The output of V is 1 if a MAC pair is from S; if it is 0, V will reject this pair.

3.1. Algorithm Definition

The AART is the protocol with the following algorithms:

- $(gpk, gsk) \overset{\$}{\leftarrow} Init(\cdot)$: it is the initialization algorithm to create group tree, generate the public group key gpk and public group key gsk.
- $(C, \sigma) \leftarrow Enc(gpk, gsk, m)$: it is the encryption algorithm to encrypt the message m with gpk and gsk. The outputs are a ciphertext C and a MAC σ.
- $m \cup \perp \leftarrow Dec(gpk, gsk, C, \sigma)$: it is the decryption algorithm to check the σ and decrypt the ciphertext C. The output is the message m if it is decrypted correctly or $\perp$ if it does not pass the validation of σ.

The sub-algorithms involved in the AART are defined as follows:

- $\{k_1, ..., k_n\} \leftarrow SKG(gpk, gsk)$: it is the session keys generation algorithm where $\{k_1, ..., k_n\} \in \mathcal{K}^n$.
- $(pos, path) \leftarrow Update(gpk, gsk, pos)$: it is the update algorithm to refresh the leaf of the sender after he encrypts a message. pos is the position of the leaf to be updated, and $path$ is the updated public key set in the group tree.
- $gpk \leftarrow UpdateGpk(gpk, pos, path)$: it is the update algorithm to replace part of the public keys of group tree according to $path$ and pos after the receiver decrypts a message.

The encryption oracle *Enc* and decryption oracle *Dec* made up of these sub-algorithms and tools are illustrated in Figure 1.

```
$AE_{\mathcal{S}}^{\mathcal{A}}(\lambda)$
$(gpk, gsk) \leftarrow_\$ Init(\cdot)$
$gpk \rightarrow \mathcal{A}$
$(m_0, m_1, c^*, \sigma^*) \leftarrow \mathcal{A}^{Enc(gpk,gsk,m)}$
$b \leftarrow_\$ \{0, 1\}$
$c_b, \sigma_b \leftarrow Enc(gpk, gsk, m_b)$
$c_b, \sigma_b \rightarrow \mathcal{A}$
$\hat{b} \leftarrow_\$ \mathcal{A}$
return $b = \hat{b}\, OR\, V(k_2(c^*, \sigma^*))$
```

```
$Enc(gpk, gsk, m)$
$k_1, k_2 \leftarrow SKG(gpk, gsk)$
$c \leftarrow E(k_1, m), \sigma \leftarrow S(k_2, c)$
return $c, \sigma$
```

Figure 1. Authenticated encryption.

3.2. Security Model

In the security models, messages queried by $\mathcal{A}$ are from $\mathcal{M}$ with the same length. In the challenge phase, the messages from $\mathcal{A}$ are different from queried messages. The adversaries mentioned in each definition are all probability polynomial time (PPT) attackers.

Unforgeability of MAC. The adversary on a MAC system attacks a chosen message and tries to forge a MAC pair that can pass the MAC system. The attacking game of unforgeability is shown in Figure 2. If $Adv_{UNF} = |Pr(V(k, m^*, \sigma^*) = 1)|$ is negligible, the MAC system can satisfy unforgeability.

Chosen Ciphertext Attack. The adversary of IND-CCA cannot only ask the plaintext encryption query but also has the ability to access decryption of the cipher. The attacking game of IND-CCA is shown in Figure 2. An encryption scheme $\mathcal{S}$ is IND-CCA if $Adv_{CCA}[\mathcal{A}, \mathcal{S}] = |\Pr(\hat{b} = b) - \frac{1}{2}|$ is negligible.

Forward Secrecy. The definition shows that the adversary cannot reveal the forward session keys when the keys are compromised. The attack game of FS is shown in Figure 2.

Oracle $\mathcal{O}$ illustrates the forward encryption. After the challenge phase, the adversary can run decryption oracle *Dec*.

```
$FS_{\mathcal{S}}^{\mathcal{A}}(\lambda)$
$(gpk, gsk) \leftarrow_{\$} Init(\cdot)$
$gpk \rightarrow \mathcal{A}$
$(m^*) \leftarrow \mathcal{A}^{\mathcal{O}(gpk,gsk,m_0^i,m_1^i)}$
$c, \sigma \leftarrow Enc(gpk, gsk, m^*)$
$c, \sigma, gsk, k_1, k_2 \rightarrow \mathcal{A}$
$\hat{b_i} \leftarrow \mathcal{A}$
return $b_i = \hat{b_i}$
```

```
$PCS_{\mathcal{S}}^{\mathcal{A}}(\lambda)$
$(gpk, gsk) \leftarrow_{\$} Init(\cdot)$
$gpk, gsk \rightarrow \mathcal{A}$
$(m_0, m_1) \leftarrow \mathcal{A}^{Enc(gpk,gsk,m)}$
$b \leftarrow_{\$} \{0, 1\}$
$c_b, \sigma_b \leftarrow Enc(gpk, gsk, m_b)$
$c_b, \sigma_b \rightarrow \mathcal{A}$
$\hat{b} \leftarrow \mathcal{A}$
return $b = \hat{b}$
```

```
$Enc(gpk, gsk, m)$
$k_1, k_2 \leftarrow SKG(gpk, gsk)$
$c \leftarrow E(k_1, m), \sigma \leftarrow S(k_2, c)$
return $c, \sigma$
```

```
$\mathcal{O}(gpk, gsk, m_0^i, m_1^i)$
$b_i \leftarrow_{\$} \{0, 1\}$
$c_b^i, \sigma_b^i \leftarrow Enc(gpk, gsk, m_b^i)$
return $c_b^i, \sigma_b^i$
```

Figure 2. Forward Secrecy and Post-Compromised Security.

An encryption scheme $\mathcal{S}$ is FS if $Adv_{FS}[\mathcal{A}, \mathcal{S}] = |\Pr(\hat{b}_i = b_i) - \frac{1}{2}|$ for any i is negligible.

Post-Compromised Secure. This definition shows that when the key is compromised after at most Q times queries, the channel will be refreshed and secure again. The attacking game of PCS is shown in Figure 2. The adversary can access the decryption oracle before and after the challenge phase. An encryption scheme $\mathcal{S}$ is PCS if $Adv_{PCS}[\mathcal{A}, \mathcal{S}] = |\Pr(\hat{b} = b) - \frac{1}{2}|$ is negligible.

Internal Group Anonymity. This definition shows that the adversary who knows the secret key cannot distinguish the identity of the target message sender. The attacking game of IGA is shown in Figure 3. An encryption scheme $\mathcal{S}$ is IGA secure if $Adv_{IGA}[\mathcal{A}, \mathcal{S}] = |\Pr(\hat{b} = b) - \frac{1}{2}|$ is negligible. After $\mathcal{C}$ receives the challenge, he should update the group tree according to the position b. If the $Update$ algorithm of a protocol cannot cut off the relation between b and the updated position, the adversary will win the game. For the example of ART, because $\mathcal{A}$ knows the updated position of the sender, it means that in this definition, $c_{b,1}$ is related to b and can be accessed by $\mathcal{A}$. So, in ART, $Adv_{IGA}[\mathcal{A}, \mathcal{S}] = 1$.

```
$IGA_{\mathcal{S}}^{\mathcal{A}}(\lambda)$
$(gpk, gsk) \leftarrow_{\$} Init(\cdot)$
$gpk, gsk \rightarrow \mathcal{A}$
$m \leftarrow \mathcal{A}, b \leftarrow_{\$} \{0, 1\}$
$c_b, \sigma_b \leftarrow Enc(gpk, gsk, m, b)$
$c_b, \sigma_b \rightarrow \mathcal{A}$
$\hat{b} \leftarrow_{\$} \mathcal{A}$
return $b = \hat{b}$
```

```
$Enc(gpk, gsk, m, b)$
$k_1, k_2 \leftarrow SKG(gpk, gsk)$
$c_b^1 \leftarrow Update(gpk, gsk, b)$
$c_b^0 \leftarrow E(k_1, m), \sigma \leftarrow S(k_2, (c_b^0, c_b^1))$
return $(c_b^0, c_b^1), \sigma_b$
```

Figure 3. External Group Anonymity.

External Group Anonymity. The security model of EGA is shown in Figure 3.

If $Adv_{EGA}[\mathcal{A}, \mathcal{S}] = |Pr(\hat{b} = b) - \frac{1}{2}|$ is negligible, an encryption scheme $\mathcal{S}$ is EGA. To make it indistinguishable, the only clue for the adversary is the output of *Enc*. It includes three parts: associated data *pos* and *path*, ciphertext c, and MAC σ. For ART and Signal, identity is an important associated data and easy to be distinguished. If an adversary cannot distinguish those associated data, it means that he cannot locate a user in an exact group.

4. Our Construction

4.1. Security Goals

Our construction aims to ensure security against the five kinds of adversaries in IND-CCA, FS, PCS, IGA, and EGA. All of the adversaries can deliver and modify the message, control the message server, and have the ability to access the decryption oracle. Except for IND-CCA, current random values including secret keys, session keys, and leaf keys can be compromised. To break the security features, the adversary can access the Key Derived Function (KDF) as a random oracle. Our construction does not consider the impersonating attack when the keys are compromised. Besides, the condition is not considered that the initial stage is compromised, and it assumes that the initial stage is based on a trusted third-party.

4.2. Security Assumption and Notation

In this subsection, the necessary assumptions and notations for AART are defined. $x \overset{\$}{\leftarrow} X$ means choosing a group element x from group X randomly. A secure pseudorandom generator (PRG) prg is to pick up the update position for group members. Sig is a secure signature, and $I = (S, V)$ is a secure MAC system. $\mathcal{E} = (E_{CPA}, D_{CPA})$ is an IND-CPA encryption scheme, $\mathbb{Z}_q$ is a finite field, q is a big prime number. The basic operation of AART is over point group $\mathbb{P}$ of Elliptic Curve (EC), where $\mathbb{P} = \{(x, y) \in \mathbb{Z}_q \times \mathbb{Z}_q : (x, y) \in EC\} \bigcup \{\infty\}$. The generator of $\mathbb{P}$ is P.

Decisional Diffie-Hellman Problem (DDHP). DDHP is to distinguish two tuples ($a \cdot P$, $b \cdot P$, $ab \cdot P$) and ($a \cdot P$, $b \cdot P$, $z \cdot P$), where $a, b \in \mathbb{Z}_q$ and $z \overset{\$}{\leftarrow} \mathbb{Z}_q$. The advantage for any PPT adversary to deal with DDHP is negligible.

Computational Diffie-Hellman Problem (CDHP). CDHP is to compute $ab \cdot P$, given a tuple ($a \cdot P$, $b \cdot P$), where $a, b \in \mathbb{Z}_q$. The advantage for any PPT adversary to deal with CDHP is negligible.

Pseudo-Random Function Oracle Diffie-Hellman (PRF-ODH) [20]. Assume a secure PRF $t(\cdot)$ is: $\mathbb{P} \to \mathbb{Z}_q$, which maps the group element of $\mathbb{P}$ to an element of $\mathbb{Z}_q$. If DDHP is held in group $\mathbb{P}$ and t is a secure PRF over $\mathbb{P}$, general PRF-ODH assumption is satisfied on $\mathbb{P}$ such that if $z \overset{\$}{\leftarrow} \mathbb{Z}_q$, given $(a \cdot P, b \cdot P, t(ab \cdot P)), (a \cdot P, b \cdot P, t(z \cdot P))$, the probability adversary distinguishes $t(ab \cdot P)$, and $t(z \cdot P)$ is negligible. Because of PRF-ODH, CDHP is still satisfied over $\mathbb{P}$ and t if $z \overset{\$}{\leftarrow} \mathbb{Z}_q$, given $(a \cdot P, b \cdot P)$, the advantage that the adversary computes $t(ab \cdot P)$ is negligible.

Node. $node$ is the basic unit of group tree. The construction of $node$ is

- $node[i]$: the ith leaf node of group tree;
- $node[i].sk$: the secret key of $node[i]$;
- $node[i].pk$: the public key of $node[i]$;
- $node[i].sibling$: the sibling of $node[i]$;
- $node[i].p$: the parent of $node[i]$.

Other operations are outlined: $push$ is to push an element to the end of a list. pop is to get and remove the first element from a list. agt is the tree of public and private keys. $size()$ is to get the number of group members or the number of a list. $KeyExchange$ can be any authentication key exchange (AKE) function or protocol. In signal, $KeyExchange$ is X3DH [5] protocol.

$$KeyExchange(ik_R, IK_I, suk_R, EK_I) = KeyExchange(ik_I, IK_R, ek_I, SUK_R)$$

This design involves several random values. The one-time secret key $node[i].sk$ is owned to user i, $node[i].pk$ is the corresponding public key. (ik, IK) is the identity key pair, (ek, EK) is the short-term key pair. ik and ek are kept by the user, and IK, EK are published. j denotes the sequence number of current stage. Session keys mk_j, r_j, ck_j are derived from $KDF(ck_{j-1}, tk_j)$. mk_j is used to encrypt message, r_j is used to calculate one-time address, and ck_j is used to generate MAC and session key pair for stage $j + 1$.

4.3. Internal Group Anonymity

4.3.1. Group Setup

Considering the three-member group, let A, B, and C be the group members. The initialization algorithm *Init* creates an anonymous group tree and sets up a communication channel. The leaves A, B, and C stand for each group member. This tree is created by the group initiator A. An overview of the group tree is shown in Figure 4.

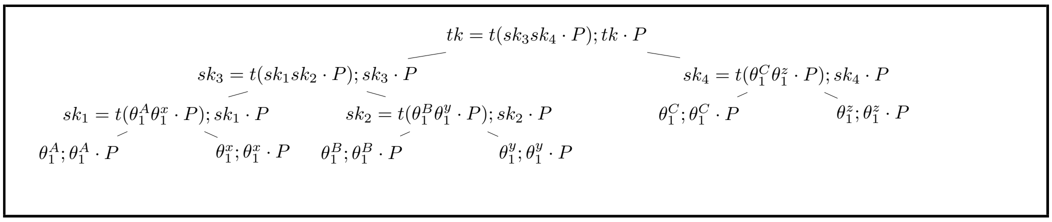
$tk = t(sk_3 sk_4 \cdot P); tk \cdot P$
$sk_3 = t(sk_1 sk_2 \cdot P); sk_3 \cdot P$
$sk_4 = t(\theta_1^C \theta_1^z \cdot P); sk_4 \cdot P$
$sk_1 = t(\theta_1^A \theta_1^x \cdot P); sk_1 \cdot P$
$sk_2 = t(\theta_1^B \theta_1^y \cdot P); sk_2 \cdot P$
$\theta_1^C; \theta_1^C \cdot P$
$\theta_1^z; \theta_1^z \cdot P$
$\theta_1^A; \theta_1^A \cdot P$
$\theta_1^x; \theta_1^x \cdot P$
$\theta_1^B; \theta_1^B \cdot P$
$\theta_1^y; \theta_1^y \cdot P$

Figure 4. Anonymous group tree overview.

The *Init* procedure is shown as follows:

- Ask for public key pairs (IK_i, EK_i) of each group member through the third channel.
- Generate setup key $suk \xleftarrow{\$} \mathbb{Z}_q^*$. Let $SUK \leftarrow suk \cdot P$. Generate A's leaf key pair $(\theta_0^A, \theta_0^A \cdot P)$ such that $\theta_0^A \xleftarrow{\$} \mathbb{Z}_q^*$. θ_0^i is the leaf secret key of user i and $\theta_0^i \cdot P$. Set initial chain key $ck_0 \xleftarrow{\$} \mathcal{K}$.
- Send IK_A, SUK, ck_0 to other group members via a trusted third-party, which means that the adversary cannot access these messages and reveal the identity of other group members in the initial session.
- Generate leaf keys of other members: $\theta_0^i \leftarrow KeyExchange(ik_A, IK_i, suk, EK_i)$, generate random leaf key as $\theta_0^i \xleftarrow{\$} \mathbb{Z}_q^*$.
- Set up group tree by $agt \leftarrow Create()$. Let the root private key and public key be (tk_1, TK_1). Set gpk as public group tree that deletes all secret keys from agt.
- Run $\sigma_0 \leftarrow Sig(ik_A, gpk_1)$ and broadcast (gpk_1, σ_0) to other group members.

Create and *Init* algorithms are illustrated in Algorithm 1.

When initiating anonymous group tree, the initiator has the full view of group tree, including the private leaf key of each node. After receiving this tree, other group members should check if (IK_A, gpk_1, σ_0) is valid or not. If σ_0 is valid, each group member will accept this tuple. He will only obtain public part gpk_1 and his private leaf key. Leaf keys can be calculated by running

$$\theta_0^i \leftarrow KeyExchange(ik_i, IK_A, ek_i, SUK) \tag{1}$$

After getting θ_0^i, group members should calculate their public leaf keys to ensure the position i of them. If the pk in gpk_1 of kth leaf is equal to $\theta_0^i \cdot P$, the position of this group member is $i \leftarrow k$. Then, he generates the group shared key tk_1 according to procedure $KeyGen(i, node[i], gpk_1)$:

1. Parent node $p \leftarrow node[i].p, s \leftarrow node[i]$
2. Find s's sibling node $s.sibling$
3. Calculate $p.sk \leftarrow t(s.sk \cdot s.sibling.pk)$
4. set $s \leftarrow p, p \leftarrow s.p$
5. If p is null, $tk \leftarrow s.sk$, else go to step 2

According to Equation (1), the group initiator knows the location of each member in gpk_1. However, each other member only knows his own location.

Algorithm 1 Anonymous Tree Generation

```
function $Create(node, size)$
    if $size \neq 1$ then
        if $size$ is odd then
            Let last node of $newNode$ be $node[size]$
        end if
        for $i = 1; i < size; i+ = 2$ do
            $newNode[(i+1)/2].sk \leftarrow t(node[i].sk \cdot node[i+1].pk)$
            $newNode[(i+1)/2].pk \leftarrow newNode[(i+1)/2].sk \cdot P$
            Let $newNode[(i+1)/2]$ be the parent of $node[i]$ and $node[i+1]$
        end for
        return $Create(newNode, size(newNode))$
    else
        return $node$
    end if
end function
procedure $Init(ik_A, IK, EK,$size $n)$
    $size \leftarrow 2n, suk \xleftarrow{\$} \mathbb{Z}_q^*, SUK \leftarrow suk \cdot P, ck_0 \xleftarrow{\$} \mathcal{K}$
    Send $IK_A, SUK, ck_0$ to other members through trust third-party
    for each $i \in [1, 2n]$ do
        if $i \, mod \, 2 = 0$ or $i = A$ then
            $node[i].sk \xleftarrow{\$} \mathbb{Z}_q^*$
        else
            $node[i].sk \leftarrow KeyExchange(ik_A, IK_i, suk, EK_i)$
        end if
    end for
    $agt \leftarrow Create(node, size), gpk \leftarrow agt$, delete all $sk$ from $gpk$
    Run $\sigma_0 \leftarrow Sig(ik_A, gpk_1)$ and broadcast $(gpk_1, \sigma_0)$ to other group members
    return $gpk, agt, node$
end procedure
```

4.3.2. Direct Updating

In order to satisfy FS and PCS, when one participant sends a message, the group tree should be updated. In stage j, the root key tk_j should be generated from gpk_j and the user's leaf secret key. After sending or receiving a message, gpk_j should be updated as gpk_{j+1}, which means that session key should be used only once. In the update phase, group members can decide to update the group tree anonymously or directly. The overview of directly updating is illustrated in Figure 5. Its procedure is described as follows (B stands for the position of the updated node):

1. Set $node[B].sk \leftarrow \theta_1^B \xleftarrow{\$} \mathbb{Z}_q^*$, $node[B].pk \leftarrow node[B].sk \cdot P$
2. Update $sk_2 \leftarrow t(\theta_1^B \theta_1^y \cdot P); pk_2 \leftarrow sk_2 \cdot P$
3. Update $sk_3 \leftarrow t(sk_1 sk_2 \cdot P); pk_3 \leftarrow sk_3 \cdot P$
4. Update $tk \leftarrow t(sk_3 sk_4 \cdot P); TK \leftarrow tk \cdot P$
5. Broadcast $B, node[B].pk, pk_2, pk_3$ to all group members

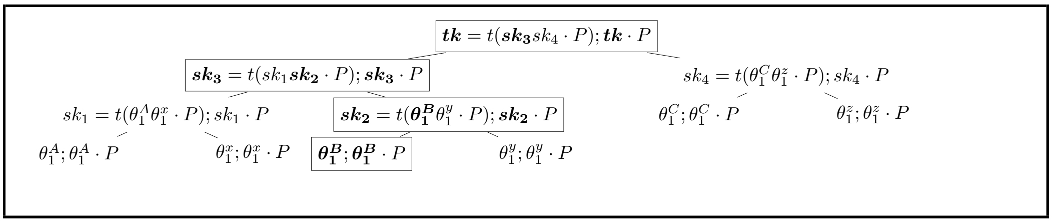
$\boldsymbol{tk} = t(\boldsymbol{sk_3} sk_4 \cdot P); \boldsymbol{tk} \cdot P$
$\boldsymbol{sk_3} = t(sk_1 \boldsymbol{sk_2} \cdot P); \boldsymbol{sk_3} \cdot P$
$sk_4 = t(\theta_1^C \theta_1^z \cdot P); sk_4 \cdot P$
$sk_1 = t(\theta_1^A \theta_1^x \cdot P); sk_1 \cdot P$
$\boldsymbol{sk_2} = t(\boldsymbol{\theta_1^B} \theta_1^y \cdot P); \boldsymbol{sk_2} \cdot P$
$\theta_1^C; \theta_1^C \cdot P$
$\theta_1^z; \theta_1^z \cdot P$
$\theta_1^A; \theta_1^A \cdot P$
$\theta_1^x; \theta_1^x \cdot P$
$\boldsymbol{\theta_1^B}; \boldsymbol{\theta_1^B} \cdot P$
$\theta_1^y; \theta_1^y \cdot P$

Figure 5. Non-anonymous updating group tree (updated nodes and values are marked in bold).

After receiving the updated public keys, others update the public keys of B and its ancestor nodes, and tk_{j+1} is derived according to *KeyGen*.

4.3.3. Anonymous Updating

Because the group initiator knows the location of each member, he can see which one is to update group tree. So, the initiator knows who sent the target message. In order to limit the authority of the initiator, the relation between the updated location and identity should be separated. By using random node, this feature can be obtained according to Figure 6. The procedure is shown as follows (b stands for the updated node's position):

1. $b \leftarrow prg(\{2, 4, 6, ..., 2n\})$
2. Set $node[b].sk \leftarrow \theta^i \xleftarrow{\$} \mathbb{Z}_q^*, node[B].pk \leftarrow node[B].sk \cdot P$
3. Update $sk_2 \leftarrow t(\theta_1^B \theta_1^y \cdot P); pk_2 \leftarrow sk_2 \cdot P$
4. Update $sk_3 \leftarrow t(sk_1 sk_2 \cdot P); pk_3 \leftarrow sk_3 \cdot P$
5. Update $tk \leftarrow t(sk_3 sk_4 \cdot P); TK \leftarrow tk \cdot P$
6. Broadcast $b, node[b].pk, pk_2, pk_3$ to all group members

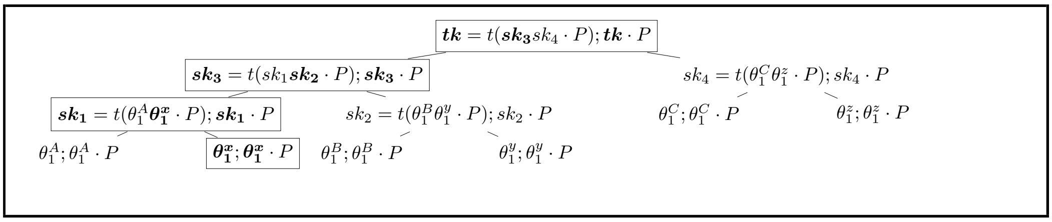
$\boldsymbol{tk} = t(\boldsymbol{sk_3} sk_4 \cdot P); \boldsymbol{tk} \cdot P$
$\boldsymbol{sk_3} = t(sk_1 \boldsymbol{sk_2} \cdot P); \boldsymbol{sk_3} \cdot P$
$sk_4 = t(\theta_1^C \theta_1^z \cdot P); sk_4 \cdot P$
$\boldsymbol{sk_1} = t(\theta_1^A \boldsymbol{\theta_1^x} \cdot P); \boldsymbol{sk_1} \cdot P$
$sk_2 = t(\theta_1^B \theta_1^y \cdot P); sk_2 \cdot P$
$\theta_1^C; \theta_1^C \cdot P$
$\theta_1^z; \theta_1^z \cdot P$
$\theta_1^A; \theta_1^A \cdot P$
$\boldsymbol{\theta_1^x}; \boldsymbol{\theta_1^x} \cdot P$
$\theta_1^B; \theta_1^B \cdot P$
$\theta_1^y; \theta_1^y \cdot P$

Figure 6. Anonymous updating group tree (updated values are marked in bold).

Because in group tree $node[i], i \in \{2, 4, 6, ..., 2n\}$ are random nodes, this means that the leaf keys of these nodes are generated randomly, and thus no group member is located in these nodes. In this way, the initiator cannot bind the sender with a random node. Therefore, he cannot reveal the identity of the sender.

4.4. External Group Anonymous Encryption

4.4.1. One-Time Address

Although ratchet tree can provide PCS and FS, it delivers messages through central servers. If those servers are controlled by the adversaries, they can know the relations of all users. With the help of the topological net, attackers can perform behavior analysis to infer the identities of the user.

One-time address applied in Monero [8] tries to hide the identity of receiver using Equation (2).

$$addr \leftarrow H(r \cdot PK_B^s) \cdot P + PK_B^v \tag{2}$$

Here, $PK_B^s \leftarrow sk_B^s \cdot P$ and $PK_B^v \leftarrow sk_B^s \cdot P$ are the long-term public keys of user Bob. $H : \mathbb{P} \leftarrow \mathbb{Z}_q$ is a collision-resistant hash function. If user Alice wants to trade with Bob, she first generates $r \xleftarrow{\$} \mathcal{K}$, calculates $addr$, and then puts $r, addr$ and transactions onto the block chain. Bob should use r and his secret key pairs to validate the $addr$. Because $addr$ is changed by r and r is randomly chosen, $addr$ is changed in each transaction. Because DDHP is hard in PRF-ODH, the adversary cannot reveal the identity of Bob from $addr$. However, because Bob should check all $addr$, the valid operation will cost a lot of time. The idea from Monero's one-time address is to hide the group public key, so that cloud servers cannot distinguish different messages from different groups according to one-time address. The SKG of our construction contains two parts: Equations (3) and (4).

$$mk_j, r_j, ck_j \leftarrow KDF(ck_{j-1}, tk_j) \tag{3}$$

$$addr_j \leftarrow H(t(r_j \cdot P)) \cdot P + tk_j \cdot P \tag{4}$$

AART generates the pseudorandom value mk_j, r_j, ck_j from tk_j and ck_{j-1} based on $KDF : \mathcal{K} \times \mathbb{Z}_q \to \mathcal{K}^3$ modeled as random oracle, so that group members can pre-calculate the one-time address for each message.

4.4.2. Encryption and Decryption

Here $type \in \{0, 1\}$ is the updated type: 0 is direct update, 1 is anonymous update.

- $SKG(node[i]_j, gpk_j, ck_{j-1})$:
  - $tk_j \leftarrow KeyGen(i, node[i]_j, gpk_j)$
  - $mk_j, r_j, ck_j \leftarrow KDF(ck_{j-1}, tk_j)$
  - $addr_j \leftarrow H(t(r_j \cdot P)) \cdot P + tk_j \cdot P$
- $Enc(node[i]_j, gpk_j, type_j, ck_{j-1})$:
  - $(mk_j, r_j, addr_j, ck_j) \leftarrow SKG(node[i]_j, gpk_j, ck_{j-1})$
  - $(pos_j, path_j, gpk_{j+1}) \leftarrow Update(i, gpk_j, type_j, node[i]_j)$
  - $c_j \leftarrow E_{CPA}(mk_j, m_j)$
  - $\sigma_j \leftarrow S(ck_j, (c_j, pos_j, path_j))$
  - $Send((c_j, pos_j, path_j, \sigma_j), addr_j, server)$
  - $output : c_j, \sigma_j, addr_j, gpk_{j+1}$
- $Dec(gpk_j, node[i], ck_{j-1})$
  - $(mk_j, r_j, addr_j, ck_j) \leftarrow SKG(node[i]_j, gpk_j, ck_{j-1})$
  - $cipher \leftarrow Get(addr_j, server)$
  - If $cipher = \perp$: output $\perp$
  - $c_j, pos_j, path_j, \sigma_j \leftarrow cipher$
  - If $V(ck_j, (c_j, pos_j, path_j), \sigma_j) \neq 1$: output $\perp$
  - else: $(m_j, pos_j, path_j) = D_{CPA}(mk_j, c_j)$
  - $gpk_{j+1} \leftarrow UpdateGpk(pos_j, path_j, gpk_j)$
  - $output : m_j, gpk_{j+1}$

$Update$ is the algorithm to update the group tree during encryption, and $UpdateGpk$ is to update the group tree after receiving updated $path$. The details of these two algorithms are illustrated in Algorithm 2. $Send(msg, addr, server)$ means putting message msg on the server according to the position of $addr$. $Get(addr, server)$ means getting the message from the position $addr$ in the server. If sending is wrong or nothing is obtained, the response of the server is $\perp$. These messages can be observed and accessed by the adversary.

Algorithm 2 Update Group Tree

```
function $Update(i, gpk_j, type_j, node_j)$
    if $type_j = 0$, $pos_j = i$, otherwise $pos_j \leftarrow prg(\{2, 4, 6, ..., 2n\})$
    $node_{j+1} \leftarrow node_j$, $node[pos_j]_{j+1}.sk \xleftarrow{\$} \mathbb{Z}_q^*$, $node[pos_j]_{j+1}.pk \leftarrow node[pos_j]_{j+1}.sk \cdot P$
    return $pos, UpdatePath(gpk_j, node_{j+1}, pos_j)$
end function
function $UpdatePath(gpk_j, node_j, pos_j)$
    $cur \leftarrow node[pos]_{j+1}$, $path_j \leftarrow [\ ]$
    while current node $cur$ is not the root do
        the $sk$ of $cur$'s parent is $t(cur.sk \cdot cur.sibling.pk)$, the $pk$ of $cur$'s parent is its $sk \cdot P$
        $path_j.push(cur.pk)$, let $cur$ move to the parent of $cur$
    end while
    return $path_j, cur$
end function
function $UpdateGpk(pos_j, gpk_j, path_j, node_j)$
    $tmp \leftarrow node[pos_j]$
    while $path_j \neq [\ ]$ do
        $tmp.pk \leftarrow path_j.pop()$, $tmp \leftarrow tmp.p$
    end while
    return $tmp$
end function
```

5. Security Analysis

In this section, it proves that AART satisfies the secure definitions of IND-CCA, FS, PCS, IGA, and EGA. The sequence of current stage is j.

5.1. IND-CCA Security

Theorem 1. *Let $\mathcal{E} \leftarrow (E_{CPA}, D_{CPA})$ be a cipher, and $I \leftarrow (S, V)$ is a MAC system. KDF : $\mathcal{K} \times \mathbb{Z}_q \rightarrow \mathcal{K}^3$ is modeled as a random oracle. Assuming $\mathcal{E}$ is IND-CPA secure and I is a secure MAC system, if adversary $\mathcal{A}$ has the advantage to break IND-CCA of AART, with Q_d times decryption queries and Q_H time Random Oracle queries, then there exists an adversary $\mathcal{B}_{UNF}$ against I, an adversary $\mathcal{B}_{PRF-ODH}$ against CDHP in PRF-ODH, and an adversary $\mathcal{B}_{CPA}$ against IND-CPA of $\mathcal{E}$ with the following bound:*

$$\begin{aligned} Adv_{CCA}^{RO}[\mathcal{A}, AART] \leq & Q_H \cdot Adv_{CDHP}[\mathcal{B}_{PRF-ODH}, \mathbb{P}] + Adv_{CPA}[\mathcal{B}_{CPA}, \mathcal{E}] \\ & + Q_d Adv_{UNF}[\mathcal{B}_{UNF}, I] \end{aligned} \tag{5}$$

Proof. In each Game$_j$, b is randomly chosen by $\mathcal{C}$, and $\hat{b}$ is the output of $\mathcal{A}$. W_j is the event that in Game$_j$, $b = \hat{b}$. The decryption query is defined in Game$_0$ as

1. When receiving c_j, σ_j from adversary, check if $V(k_0, c_j, \sigma_j) = 1$.
2. If it is true, reply $D(k_1, c_j)$, else $\perp$.

It should prove that

$$Adv_{CCA}^{RO}[\mathcal{A}, AART] = |\Pr(W_0) - \frac{1}{2}| \tag{6}$$

Then, Game$_0$ is changed into Game$_1$. Step 1 is deleted and step 2 is changed to send "reject" except when $j = \omega \in \{1, Q_d\}$. It can be seen that the difference between Game$_0$ and Game$_1$ is the event that c_ω is queried. According to the definition of Unforgeability, there is

$$Adv_{UNF}[\mathcal{B}_{UNF}, I] = |\Pr(W_0) - \Pr(W_1)|/Q_d \tag{7}$$

To simplify, we will remove the decryption query in accordance with Equation (7) from our proofs. Thus, Game$_1$ is the IND-CPA game of AART and then is modified into Game$_2$.

The random oracle is recorded by MAP. Game$_2$ is the same as Game$_1$ except for deleting MAP operation of step 8 from Game$_1$. Event $\mathcal{Z}$ is defined such that $\mathcal{A}$ queries tk_{Q_1+1}, ck_{Q_1+1} in domain(MAP). The difference between these two games is that event $\mathcal{Z}$ happens. So there is

$$|\Pr(W_2) - \Pr(W_1)| = \Pr(\mathcal{Z}) \tag{8}$$

Using CDHP. If event $\mathcal{Z}$ happens, it means that $\mathcal{A}$ queries $tk_{Q_1+1}, ck_{Q_1+1} \in$domain ($MAP$), which can be used to break CDHP and to construct $\mathcal{B}_{PRF-ODH}$. To break CDHP, one tk, ck pair should be picked out, but $\mathcal{B}_{PRF-ODH}$ is not sure which one in domain(MAP) is the right answer. Assume there are at most Q_2 times random oracle queries; the probability to select right pair is at most $\frac{\Pr(\mathcal{Z})}{Q_2}$. We use Game$_2$ to construct Game$_{CDHP}$. Instead of running $Init, KeyGen, Update$, $\mathcal{B}_{PRF-ODH}$ should query them from $\mathcal{C}_{PRF-ODH}$. The gray parts with boxes of Game$_2$ challenger are constructed as $\mathcal{C}_{PRF-ODH}$. Thus, from $\mathcal{A}$'s view, there is no difference between Game$_2$ and Game$_{CDHP}$. Event $\mathcal{Z}$ happens $\iff$ $tk_{Q_1}, ck_{Q_1} \in$ domain(MAP) when $\mathcal{B}_{PRF-ODH}$ finishes the game. Let $Q \leftarrow Q_2$, because the pairs may be queried more than once, the size of domain(MAP) is no greater than Q. So, there is

$$Adv_{CDHP}[\mathcal{B}_{PRF-ODH}, \mathbb{P}] \geq \frac{\Pr(\mathcal{Z})}{Q} \tag{9}$$

According to Game$_2$, to deal with $\Pr(W_2)$ means to deal with IND-CPA. So

$$|\Pr(W_2) - \frac{1}{2}| = Adv_{CPA}[A, AART] \tag{10}$$

Using CPA. Game$_{CPA}$ can be constructed from Game$_2$. Let Game$_2$ challenger be $\mathcal{B}_{CPA}$ except that after receiving message from $\mathcal{A}$, $\mathcal{B}_{CPA}$ should run encryption query to $\mathcal{C}_{CPA}$ such like the gray parts with no boxes in Figure 7. So there is

$$Adv_{CPA}[\mathcal{A}_{CPA}, AART] = Adv_{CPA}[\mathcal{B}_{CPA}, E] \tag{11}$$

Combining Equations (6)–(11), Theorem 1 can be derived. Because CDHP in PRF-ODH is hard and E is IND-CPA cipher, I is secure MAC system, $\mathcal{A}$ cannot win Game$_0$. So $Adv_{CCA}[\mathcal{A}, AART]$ is negligible. IND-CCA of AART is satisfied. □

5.2. Forward Secrecy

Theorem 2. *Let $KDF : \mathcal{K} \times \mathbb{Z}_q \to \mathcal{K}^3$ be modeled as a random oracle. When the keys of stage $j+1$ are leaked, if adversary $\mathcal{A}$ can break FS of AART, there exists adversary $\mathcal{B}_{CCA}$ that can break the IND-CCA of stage j with the advantage:*

$$Adv^{RO}_{FS}[\mathcal{A}, AART] \leq Q \cdot Adv^{RO}_{CCA}[\mathcal{A}, AART] \tag{12}$$

Proof. Assume there are Q stages. According to SKG and $Update$, tk_j is derived from gpk_j, and session keys of stage j are generated by tk_j, ck_{j-1}. So if all random values including sk of each user, tk_j, session keys mk_j, r_j, ck_j are compromised, and adversary $\mathcal{A}$ wants to get session keys of stage $j-1$, he needs to know tk_{j-1}. If the current leaf key of each user is not compromised, each stage can be reduced to an IND-CCA game in Theorem 1. If the current leaf key is compromised, he can get tk_{j-1} when the leaf key is not updated. So he can try to get ck_{j-2} to break FS. In order to get ck_{j-2}, he should get ck_{j-3} recursively until the initial stage. However, the initial stage is run through secure AKE and a trusted third-party, and the adversary cannot break FS through this way. Assume challenger $\mathcal{C}$ is the group creator. Game$_0$ is illustrated in Figure 8.

```
Game$_1$ $\boxed{\text{Game}_2}$
$gpk_0, agt_0, node_1, SUK \leftarrow Init(ik_{\mathcal{C}}, IK, EK, n)$ $\boxed{\text{Run } Init \text{ query}}$
$ck_0 \overset{\$}{\leftarrow} \mathcal{K}$, set $MAP : \mathcal{K} \times \mathbb{Z}_q \rightarrow \mathcal{K}^3$, set $pos_u \overset{\$}{\leftarrow} \{1, 3, ..., 2n-1\}$
Send public key tree $gpk_0$ to $\mathcal{A}$
Plaintext Query from $i = 1$ to $Q_1$:
$m_i \leftarrow \mathcal{A}$
$tk_i \leftarrow KeyGen(pos_u, node[pos_u]_i, gpk_{i-1}), t \overset{\$}{\leftarrow} \{0, 1\}$ $\boxed{\text{Run } KeyGen \text{ query}}$
$mk_i, r_i, ck_i \leftarrow \mathcal{K}^3$
$MAP[ck_{i-1}, tk_i] \leftarrow mk_i, r_i, ck_i$ $\boxed{\text{Delete}}$
$c_i = E_{CPA}(mk_i, (m_i))$
$addr_i = H(t(r_i \cdot P)) \cdot P + tk_i \cdot P$
If $t = 0$: $pos_{i-1} \leftarrow pos_u$, otherwise $pos_{i-1} \leftarrow prg\{2, 4, ..., 2n\}$
$(pos_i, path_i, gpk_i) \leftarrow Update(pos_{i-1}, gpk_{i-1}, t, node_i)$ $\boxed{\text{Run } Update \text{ query}}$
$\sigma_i \leftarrow S(ck_i, (c_i, pos_i, path_i))$
Send $c_i, pos_i, path_i, \sigma_i, addr_i$ to $\mathcal{A}$
Challenge Query:
$m_{Q_1+1,0}, m_{Q_1+1,1} \leftarrow \mathcal{A}$
$b \overset{\$}{\leftarrow} \{0, 1\}$, run plaintext query for $m_{Q_1+1,b}$
Random Oracle Query from $j = 1$ to $Q_2$:
$\hat{tk}_j, \hat{ck}_j \leftarrow \mathcal{A}$
If $(\hat{tk}_j, \hat{ck}_j)$ in domain($MAP$): $MAP[\hat{tk}_j, \hat{ck}_j] \rightarrow \mathcal{A}$
Otherwise: $MAP[\hat{tk}_j, \hat{ck}_j] \overset{\$}{\leftarrow} \mathcal{K}^3, MAP[\hat{tk}_j, \hat{ck}_j] \rightarrow \mathcal{A}$
$\hat{b} \leftarrow \mathcal{A}$
```

Figure 7. Game$_1$ Challenger and Game$_2$ Challenger for IND-CPA.

For the ith message query, if $\hat{b}_i = b_i$, $\mathcal{A}$ wins Game$_0$. By querying each session key, root key, and plaintext encryption from the IND-CCA challenger of Game$_0$ in Figure 7, Game$_0$ can be changed into Game$_{CCA,i}$ for each stage i. According to Theorem 1:

$$Adv_{FS}^{RO}[\mathcal{B}_{FS,i}, s_i] \leq Adv_{CCA}^{RO}[\mathcal{A}, AART] \tag{13}$$

There are Q times of Game$_i$, so Theorem 2 proves to be true. Because $Adv_{CCA}^{RO}[\mathcal{A}, AART]$ is negligible, $Adv_{FS}^{RO}[\mathcal{A}, AART]$ is negligible too. Forward Secrecy of AART is satisfied. □

```
Game$_0$
Stage 0:
$gpk_0, agt_0, node_1, SUK \leftarrow Init(ik_{\mathcal{C}}, IK, EK, n)$, send $gpk_0$ to $\mathcal{A}$, $ck_0 \leftarrow \mathcal{K}$
Set $MAP : \mathcal{K} \times \mathbb{Z}_q \rightarrow \mathcal{K}^3$
Stage $i$ from $i = 1$ to $Q$:
$tk_i \leftarrow KeyGen(pos_{\mathcal{C}}, node[pos_{\mathcal{C}}]_i, gpk_{i-1})$
$mk_i, r_i, ck_i \xleftarrow{\$} \mathcal{K}^3$
$MAP[ck_{i-1}, tk_i] \leftarrow mk_i, r_i, ck_i$
Receive $m_{i,0}, m_{i,1}$ from $\mathcal{A}$, where $|m_{i,0}| = |m_{i,1}|$ and $m_{i,0}, m_{i,1} \in \mathcal{M}$
$b_i, type_i \xleftarrow{\$} \{0,1\}^2$
$c_{i,b} = E_{CPA}(mk_i, m_{i,b})$
$addr_i = H(t(r_i \cdot P)) \cdot P + tk_i \cdot P$
$pos_i, path_i, gpk_i \leftarrow \quad Update(pos_{\mathcal{C}}, gpk_{i-1}, type_i, node_{i-1})$
$\sigma_i \leftarrow MAC(ck_i, (c_{i,b}, pos_i, path_i))$
Send $c_{i,b}, pos_i, path_i, \sigma_i$ to $\mathcal{A}$
Decryption Query: directly reply $Dec$ using current stage session keys. (Actually, it would be $\perp$)
Key compromised phase(stage $Q + 1$):
Send $tk_{Q+1}$,$ik_C$,$ck_Q$, $mk_{Q+1}$,$r_{Q+1}$,$ck_{Q+1}$ to $\mathcal{A}$
Decryption Query: directly reply $Dec$ using current stage session keys. (Actually, it would be $\perp$)
$\mathcal{A}$ outputs $\hat{b}_i$, $i$ from 1 to $Q$
```

Figure 8. Game$_0$ Challenger for FS.

5.3. Post-Compromised Security

PCS is proved with Theorem 3.

Theorem 3. *Let $KDF : \mathcal{K} \times \mathbb{Z}_q \rightarrow \mathcal{K}^3$ be modeled as a random oracle. When the keys of stage j are compromised, if in the challenge stage all leaf keys are updated, the advantage of adversary $\mathcal{A}$ to break PCS of AART is equal to the advantage of $\mathcal{A}$ to break IND-CCA of stage $j + 1$, such that*

$$Adv^{RO}_{PCS}[\mathcal{A}, AART] = Adv^{RO}_{CCA,j+1}[\mathcal{A}, AART] \tag{14}$$

Proof. When other keys except for ck_j of jth session are compromised, because the keys of the next session $j + 1$ are based on ck_j, the adversary cannot derive them. So, the only way for the adversary is to break the IND-CCA of $j + 1$ session. Thus, Theorem 3 can be reduced. When all keys are compromised, if the leaf keys adversary holds are not updated until the Q session finished, the advantage for the adversary is 1. However, when each leaf key of the group tree is updated, the advantage of $\mathcal{A}$ is reduced to the IND-CCA of Qth session and becomes negligible. □

5.4. Internal Group Anonymity

IGA of AART is proven with Theorem 4.

Theorem 4. *Let KDF be modeled as random oracle, E_{CPA} be IND-CPA cipher, and prg be secure PRG; if there exists adversary $\mathcal{A}$ to break IGA, then there exists adversary $\mathcal{B}$ that breaks PRG:*

$$Adv_{IGA}[\mathcal{A}, AART] = |\Pr(b = 0) - \Pr(\hat{b} = 0)| = Adv_{PRG}[\mathcal{B}, prg] \tag{15}$$

Proof. Because the random leaf to be used in the anonymous update is chosen randomly by secure PRG, if the adversary can distinguish between two anonymous users from each other depending on their updated messages, he can break the security of PRG. □

5.5. External Group Anonymity

Theorem 5. *Let H be a collision-resistant hash function and KDF be modeled as random oracle; if adversary $\mathcal{A}$ can break EGA of AART, there exists adversary $\mathcal{B}_{PRF-ODH}$ against DDHP in PRF-ODH with the advantage:*

$$Adv_{EGA}[\mathcal{A}, AART] \leq 2 \cdot Adv_{DDHP}[\mathcal{B}_{PRF-ODH}, \mathbb{P}] \tag{16}$$

Proof. Illustrated as Figure 9, Game$_{EGA}$ includes two parts Game$_0(0)$ and Game$_0(1)$ simulating two groups. Challenger $\mathcal{C}$ plays Game$_0(b)$ with adversary $\mathcal{A}$ where $b \xleftarrow{\$} \{0,1\}$. $\mathcal{A}$ should distinguish which game is played. If the output of $\mathcal{A}$ is $\hat{b}$ and $\hat{b} = b$, $\mathcal{A}$ win Game$_{EGA}$. For each Game$_0(b)$, a DDHP game can be constructed such that tk_b is generated from random as Game$_1(b)$. W_0^b denotes that Game$_0(b)$ is played and W_1^b denotes that Game$_1(b)$ is played. According to the definition of EGA, there is

$$Adv_{EGA}[\mathcal{A}, AART] = |\Pr(W_0^0) - \Pr(W_0^1)| \tag{17}$$

According to the definition of DDHP in PRF-ODH, there is

$$\begin{aligned} Adv_{DDHP}[\mathcal{B}_{PRF-ODH}, \mathbb{P}] = |\Pr(W_0^b) - \Pr(W_1^b)| = |\Pr(W_0^b) - \frac{1}{2}| \\ 2Adv_{DDHP}[\mathcal{B}_{PRF-ODH}, \mathbb{P}] \geq |\Pr(W_0^0) - \Pr(W_0^1)| = Adv_{EGA}[\mathcal{A}, AART] \end{aligned} \tag{18}$$

Then, Theorem 5 proves to be true. Because DDHP is hard in PRF-ODH, Adv_{EGA} is negligible. So EGA of AART is satisfied. □

```
Game_0(b) [Game_1(b)]
$gpk_b, agt_b, node_b, SUK_b \leftarrow Init(ik_{\mathcal{B}}, IK_b, EK_b, n)$
$tk_b \leftarrow KeyGen(pos_b, node[pos_b], gpk_b)$ $\boxed{tk_b \xleftarrow{\$} \mathcal{Z}_q}$
$ck_b \xleftarrow{\$} \mathcal{K}$
$mk, r, ck_{new} = KDF(ck_b, tk)$
Replace the root public key of $gpk_b$ as $tk \cdot P$
$gpk_b \rightarrow \mathcal{A}$
$m \in \mathcal{M} \leftarrow \mathcal{A}$:
$addr = H(t(r \cdot P)) \cdot P + tk \cdot P$
$addr \rightarrow \mathcal{A}$
$\mathcal{A}$ outputs $\hat{b}$
```

Figure 9. Game$_0$ and Game$_1$ Challengers for EGA.

6. Discussion

We further discuss the performance and some issues when running AART.

Performance. The performance comparison can be seen from Table 1. For n group members, the number of nodes of ART is $2n$. The amount of nodes in AART is $4n$ because of the additional random nodes. Thus, the exponentiation times and storage cost to generate the public tree of AART are two times as ART. Also, the height of the group tree will be $log(2n) + 1$ in AART, which is increased by one compared with $log(n) + 1$ in ART. The complexity and storage in update phase will retain the same relationship of the heights. Moreover, there is an additional *addr* in AART. Above all, the complexity and storage of AART are close to ART.

Table 1. Performance comparison. n denotes the group size, each key exchange operation will access the exponentiation one time. Each key exchange, exponentiation, and encryption will cost one storage.

		#Exponentiation Times		**#Encryption Times**		**#Communication Storage**		**#Computation Storage**	
		Sender	**per Other**	**Sender**	**per Other**	**Sender**	**per Other**	**Sender**	**per Other**
sender keys	setup	n	n	n	n	n	n	$2n$	$2n$
	ongoing	0	0	1	1	1	1	1	1
pair-wise Signal	setup	n	n	0	0	n	n	n	n
	ongoing	n	1	$n-1$	1	n	1	$2n$	2
ART	setup	$2n$	$log(n)$	0	0	$3n-1$	$3n-1$	$2n$	$log(n)$
	ongoing	$log(n)$	$log(n)$	1	1	$log(n)+1$	$log(n)+1$	$log(n)+1$	$log(n)+1$
Ours	setup	$4n$	$log(2n)$	0	0	$5n-1$	$4n+1$	$4n$	$log(2n)$
	ongoing	$log(2n)$	$log(2n)$	1	1	$log(2n)+1$	$log(2n)+1$	$log(2n)+1$	$log(2n)+1$

For the exponentiation times, it will be $4n$ for the sender in AART because of the tree structure. Because of the Update algorithm, the time cost in the following stage will be $log(2n)$. The sender of the pair-wise Signal should update all of the channels with others. Thus, it will cost n, worse than AART. "Sender keys" will not refresh their channels, it will be 0.

For encryption times, only "sender keys" will encrypt the message keys for others. For all of these protocols, there will be only one encryption operation in each stage.

For communication storage, the sender of AART should store the $n-1$ long-term public keys of others and broadcast the $4n$ public key pairs to others; it will be $5n-1$. Each group member should not know the long-term public keys of other group members except for the creator, the cost will be $4n+1$. In ART, each member should get the identity keys of others. The ongoing cost will be $log(2n)$ because of the outputs of the Update operation. "Sender keys" will cost n for sending keys at the beginning, but it is only 1 ongoing since the ciphertext for each member is the same. According to one-to-all channels, it will take up n for both sender and others through pair-wise Signal. In the following sessions, it will cost n to refresh all channels between the sender and receivers. The computation storage is the addition of storage spent on exponentiation and encryption. It can be seen that the cost of AART at the setup stage is the largest. However, because of the tree structure, AART is more efficient in the ongoing stages compared with pair-wise Signal.

Although iMessage provides E2EE features, it cannot resist against the CCA [9] level attacker. LINE applies E2EE, but it cannot achieve FS and PCS. Tor is not an E2EE protocol because the last node of Tor knows the plaintext of the sender. ART is the first group protocol applying PCS, but it cannot cope with identity protection. With the help of the additional cost, AART can achieve FS, PCS, and anonymity at the same time compared with other protocols. The security comparison can be seen in Table 2.

About trusted third-parties. In ART, there is no efficient way to protect the initial stage from being attack. If the first is compromised, it means that all of the users' long-term secret keys can be access, and the identities of group members will be obtained at the beginning. We follow this setting, and we initialize the first stage session key by tk_1 and ck_0. The later ck_j is generated by former root key tk_j and ck_{j-1}. Thus, the ck_0 should be either empty or decided by the group creator. If ck_0 is empty, the FS cannot be satisfied when tk_j is compromised. The details can be found in the proof of Theorem 2.

Table 2. Security comparison.

Apps or Protocols	E2EE	FS	PCS	EGA	IGA
iMessage	Yes	Yes	No	No	No
LINE	Yes	No	No	No	No
Signal	Yes	Yes	No	No	No
ART	Yes	Yes	Yes	No	No
Tor and ToK	No	No	No	Yes	Yes
KEM/DEM	Yes	No	No	Yes	No
QQ and WeChat	Unknown	Yes	No	Unknown	Unknown
Ours AART	Yes	Yes	Yes	Yes	Yes

Anonymity when the key is compromised. From Theorems 2 and 3, AART can provide FS and PCS. However, it should be considered whether AART will still satisfy IGA and EGA when the key is compromised. In IGA, the adversary can be seen as the group creator, according to Theorem 4, the adversary cannot distinguish the identities of the senders even when he knows all of their secret keys. For the EGA adversary, if the key is compromised, he may know the identities when the identity keys are leaked. According to Theorem 5, he at least cannot reveal who sends the target message.

IP address. Message server may bind the IP addresses with users who access the same *addr* in the server. To avoid this situation, users can visit the server through a proxy. According to Tok, the out point of Tok should know the IP address of users. This situation cannot be avoided. However, AART just concerns the *addr* in the server. If the proxy is not controlled by the adversary and message server, or the proxy IP address is changed all the time, the adversary cannot bind the IP of users with the same group. Thus, the adversary cannot reveal the real relation of the group members in the real world.

Message conflict. In the real network environment, group members may send messages at the same time but generate different gpk of next stage, which will cause conflict and break the protocol. In AART however, all users of the same group will generate the same $addr$. If $addr$ exists, it means that the updating operation is out of date and the message should be re-encrypted again. To avoid the adversary or server taking up the *addr* of the current stage, the sender can check the MAC of *addr*. If it is wrong, this *addr* is still available for the group. When the key is compromised, AART cannot avoid the situation that the adversary generates the same *addr* and legal MAC value. However, this ability belongs to the active attacker, and we aim to prevent the adversary to become an active attacker.

Message recovery and chosen ciphertext attack. To recover messages of a group, group members should keep their *sk* of all stages along with the initial gpk_1. According to gpk_1 and sk_1, users can generate $addr_1$ and get the correct message matched by $addr_1$. Thus, users can update the correct gpk_2, while they also hold sk_2. That means all messages can be put in the server and can be recovered correctly.

Keeping all *sk* will weaken the security of AART. For IND-CCA, the challenge can reject the decryption query because the structure of AART is also a ratchet and can only be pushed forward but not in the backward direction. So, users only own the secret key of the current stage, ideally, and the former stage for the consideration of message conflict. Besides the definitions that the adversary cannot inquire about the messages in plaintext query, if the adversary can access old information and ask for decryption, the challenge will reject this request because the MAC key of old information has been deleted, and the probability that these two keys are the same is negligible. However, if users store the past secret keys, the challenge should set up a table to combine the secret keys with old messages. When queried by the IND-CCA adversary in this situation, the challenge should look for the table and decrypt the message if the requested message is matched. Therefore, in a message recovery situation, AART cannot resist the IND-CCA adversary. If IND-CCA is required, the message recovery should be given up.

Malicious group member. Malicious users who want to compromise keys or combine two group trees are included in without the help of the leaked keys. For the former situation, because of FS, PCS, IGA, and EGA, messages, as well as identities, can be protected. For the latter, although a malicious user can replace his leaf key in group A with the root of another group B, since the chain keys are different in two groups, members of group A cannot get the $addr$ of B. Therefore, the two groups cannot be combined.

About collusion attacks, in a group of n members, if there are $n - 1$ members in collusion, including the creator and the rest sending a message, they can reveal the identity of him. However, if the creator is trustworthy, collusion attackers can only know that one member sent a message, but they cannot reveal the identity of him because they do not know the long-term public key of the sender.

Dynamic group member and device. It is easy to add a new group member through $KeyExchange$. The initial leaf key can be obtained by the creator, and then the creator creates a three-node agt with one root, a new member leaf, and a new random leaf. Then the creator inserts the three-node agt to the current agt to be a complete binary tree (two leaves and their parent are thought to be one unit). The creator uses tk and three nodes agt's root public key to generate new agt's root tk and public TK. Finally, he publishes the new gpk of agt. Deleting a member is easy as well. Consider one unit as a three-node agt including a user leaf, its sibling random leaf, and their parent node, the sibling of one unit has the same parent node with this unit. To remove one user, the creator should replace the parent of the unit where the target user is located with the sibling unit, use the random leaf in the sibling unit to update the agt, and publish the new gpk to all group members.

Regarding the dynamic device, the user can share tk and ck with multiple devices, create a subtree, and let the root of the subtree replace the user leaf. When updating agt, the user should update this subtree and output the path except for the path in this subtree to group members. Then, other group members will believe that they are chatting with a multi-device user.

7. Conclusions

In this paper, we propose a multi-stage anonymous group messaging protocol called AART, which is based on the design of ART. It can provide anonymity features including IGA and EGA, while it retains the previous features such as FS and PCS of ART. The security of AART is analyzed formally. Finally, we discuss the performance of AART compared with ART, pair-wise Signal, and "sender keys" protocols as well as other problems that may exist in AART and the related solutions to them. In our future work, the effort will be focused on how to limit anonymity by tracing the secret keys and revealing the identity of malicious users.

Author Contributions: Conceptualization, K.C. and J.C.; methodology, K.C.; validation, J.C. and J.Z.; formal analysis, K.C. and J.Z.; writing—original draft preparation, K.C.; writing—review and editing, K.C. and J.C.; supervision, J.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partly supported by the National Natural Science Foundation of China under Grant No.61702212 and the Fundamental Research Funds for the Central Universities under Grant No.CCNU19TS017.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Naor, M.; Yung, M. Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, 13–17 May 1990; Ortiz, H., Ed.; ACM: New York, NY, USA, 1990; pp. 427–437.
2. Menezes, A.J.; Katz, J.; Van Oorschot, P.C.; Vanstone, S.A. *Handbook of Applied Cryptography*; CRC Press: Boca Raton, FL, USA, 1996; pp. 496.
3. Cohn-Gordon, K.; Cremers, C.; Garratt, L. On capitalisewordsPost-compromise Security. In Proceedings of the 2016 IEEE 29th Computer Security Foundations Symposium (CSF), Lisbon, Portugal, 27 June–1 July 2016; pp. 164–178.
4. Cohn-Gordon, K.; Cremers, C.; Garratt, L.; Millican, J.; Milner, K. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 1802–1819.
5. Cohn-Gordon, K.; Cremers, C.; Dowling, B.; Garratt, L.; Stebila, D. A formal security analysis of the signal messaging protocol. In Proceedings of the 2017 IEEE European Symposium on Security and Privacy (EuroS&P), Paris, France, 26–28 April 2017; pp. 451–466.
6. Turton, W.; Scigliuzzo, D. Facebook Sues Israel's NSO on Alleged WhatsApp Malware Hack. 2019. Available online: https://www.bloomberg.com/news/articles/2019-10-29/facebook-sues-israel-s-nso-over-alleged-whatsapp-malware-attack (accessed on 29 October 2019).
7. Chen, K.; Chen, J. Anonymous End to End Encryption Group Messaging Protocol Based on Asynchronous Ratchet Tree. In *International Conference on Information and Communications Security*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 588–605.
8. Sun, S.F.; Au, M.H.; Liu, J.K.; Yuen, T.H. Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In *European Symposium on Research in Computer Security*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 456–474.
9. Garman, C.; Green, M.; Kaptchuk, G.; Miers, I.; Rushanan, M. Dancing on the lip of the volcano: Chosen ciphertext attacks on apple imessage. In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016; pp. 655–672.
10. Apple Inc. iOS Security Guide. 2018. Available online: https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf (accessed on 10 September 2019).
11. LINE Inc. *Encryption Whitepaper*. 2016. Available online: https://scdn.line-apps.com/stf/linecorp/en/csr/line-encryption-whitepaper-ver1.0.pdf (accessed on 12 September 2019).
12. Isobe, T.; Minematsu, K. Breaking Message Integrity of an End-to-End Encryption Scheme of LINE. In *European Symposium on Research in Computer Security*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 249–268.
13. Borisov, N.; Goldberg, I.; Brewer, E. Off-the-record communication, or, why not to use PGP. In Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, Washington, DC, USA, 28 October 2004; pp. 77–84.
14. Tencent. Weixin Privacy Protection Guidelines. 2019. Available online: https://weixin.qq.com/cgi-bin/readtemplate?lang=en&t=weixin_agreement&s=privacy&cc=CN (accessed on 12 January 2020).
15. Tencent. Tencent Privacy Protection Platform. 2019. Available online: https://privacy.qq.com/ (accessed on 12 January 2020).
16. Dingledine, R.; Mathewson, N.; Syverson, P. *Tor: The Second-Generation Onion Router*; Technical Report; Naval Research Lab: Washington, DC, USA, 2004.
17. Tok. Tok White Paper v1.1. 2020. Available online: https://www.tok.life/static/d/TOK_WP_en.pdf (accessed on 12 January 2020).
18. Emura, K.; Kanaoka, A.; Ohta, S.; Takahashi, T. Building secure and anonymous communication channel: Formal model and its prototype implementation. In Proceedings of the 29th Annual ACM Symposium on Applied Computing, New York, NY, USA, 24–28 March 2014; pp. 1641–1648.
19. Emura, K.; Kanaoka, A.; Ohta, S.; Takahashi, T. Establishing secure and anonymous communication channel: KEM/DEM-based construction and its implementation. *J. Inf. Secur. Appl.* **2017**, *34*, 84–91. [CrossRef]
20. Brendel, J.; Fischlin, M.; Günther, F.; Janson, C. Prf-odh: Relations, instantiations, and impossibility results. In *Annual International Cryptology Conference*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 651–681.