國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master’s Thesis

支援發送者匿名性和選擇性訊息存取的連續群組密鑰協議協定

A Continuous Group Key Agreement Protocol Supporting Sender Anonymity and Selective Message Access

林義閔

I-Min Lin

指導教授: 蕭旭君 博士

Advisor: Hsu-Chun Hsiao, Ph.D.

中華民國 113 年 7 月

July, 2024

doi:10.6342/NTU202401660

國立臺灣大學碩士學位論文
口試委員會審定書

MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

支援發送者匿名性和選擇性訊息存取的連續群組密鑰協議協定

A Continuous Group Key Agreement Protocol Supporting Sender Anonymity and Selective Message Access

本論文係林義閔君（學號 R11922030）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 113 年 6 月 25 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 25 June 2024 have examined a Master's thesis entitled above presented by LIN, I-MIN (student ID: R11922030) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

蕭旭君　　[illegible]

（指導教授 Advisor）

系主任/所長 Director: 陳祝嵩

國立臺灣大學碩士學位論文
口試委員會審定書
MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

支援發送者匿名性和選擇性訊息存取的連續群組密鑰協議協定

A Continuous Group Key Agreement Protocol Supporting Sender Anonymity and Selective Message Access

本論文係林義閔君（學號 R11922030）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 113 年 6 月 25 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 25 June 2024 have examined a Master's thesis entitled above presented by LIN, I-MIN (student ID: R11922030) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

蕭旭君 ____________________ [illegible]
（指導教授 Advisor）

系主任/所長 Director: 陳祝嵩

誌謝

首先，我想感謝蕭老師兩年半來的指導。感謝老師耐心地引導我這個剛開始不懂如何做研究的學生，從雛形做到今天的成果。雖然過程並不容易，但我在其中學到的知識、方法、以及思考方式都非常有價值。也感謝老師提供的各種機會，如擔任助教、參加研討會、出國進行研究等，這些經驗大大開闊了我的視野，讓我有機會嘗試過去認為不可能的事，對我的論文和個人成長都非常有幫助。同時，感謝黎老師與游老師在百忙之中擔任我的口試委員，老師們的建議和提問都讓這篇論文更臻完善。

此外，感謝 Allen 共同耕耘這個研究主題。如果沒有你一開始提出的研究方向以及過去兩年的深入討論和投稿準備，就不會有今天的成果。也感謝以安、惟平，以及顯恩在安全性定義及證明上提供的寶貴想法。感謝 NSLab 的每一位成員。

最後感謝家人以及 Mo 在求學路上的支持與陪伴。

林義閔 謹啟

中華民國 113 年 7 月

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

摘要

群組通訊應用程式重度依賴群組密鑰協定協議以確保安全性，但這些協議的隱私方面尚未被充分探討。當不受信任的第三方應用程式，例如聊天機器人，被整合進群組通訊時，這種疏忽可能會嚴重危及使用者隱私。本文旨在通過設計一種支持發送者匿名和選擇性訊息存取的群組密鑰協定協議，來提高群組通訊應用程式的隱私保證。我們首先在威脅模型中考慮不受信任的第三方應用程式，然後基於 IETF MLS 群組通訊標準，我們提出了一個捕捉這兩個隱私特性的安全模型。此外，我們基於 MLS 標準使用的 TreeKEM 密鑰協定協議提出了一個可行的實現方法。我們的方法造成的額外計算負擔不會隨著使用者數量增加，對聊天機器人數量則是線性成長，而模組化的設計使其便於整合進 MLS 標準。

關鍵字：安全群組訊息傳輸、連續群組密鑰協議、發送者匿名性、群取控制、基於樹結構的群組密鑰管理協議

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

Abstract

Group messaging applications rely heavily on group key agreement protocols to ensure security, but the privacy aspects of these protocols have been underexplored. This oversight becomes particularly critical when untrusted third-party applications, the chatbots, are integrated into group chats, potentially compromising privacy. This paper aims to improve the privacy guarantee of group messaging applications by designing a group key agreement protocol that supports sender anonymity and message access control. We first consider untrusted third party applications in our threat model, then, based on the security model of the IETF MLS standard for group messaging, we propose a security model that captures the two privacy features. Furthermore, we propose a construction based on TreeKEM, the key agreement protocol used by MLS. Our construction imposes an overhead that is constant with respect to the number of users and linear with the number of chatbots, and the modular design makes it easy to integrate into the MLS standard.

Keywords: secure group messaging, continuous group key agreement, sender anonymity, access control, tree-based group key agreement

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

Contents

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

List of Figures

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

List of Tables

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

Chapter 1 Introduction

Group messaging applications are widely used in modern communication and are involved in various aspects of our daily lives. Platforms like WhatsApp, Messenger, and LINE are commonly used for personal interactions between family and friends, while Slack and Microsoft Teams are preferred for professional and workplace collaboration. Similarly, applications like Telegram and Discord are popular for building community connections. These platforms facilitate the transmission of messages that can include opinions, locations, photos, and confidential information associated with an individual. Given the sensitivity of this data, it is critical to ensure robust protection against access by unintended parties, including the messaging service providers. This need underscores the importance of securing group messaging to maintain user privacy and confidentiality in these popular applications.

Securing messaging services require dedicated key agreement protocols. Although technologies such as Transport Layer Security (TLS) are widely used to secure communication, they are not sufficient to secure messaging services. TLS primarily protects messages from external observers on the network, but not the messaging service providers themselves. It only establishes secure channels at the network level between communicating parties, not directly between users at the application level. To address this limitation, the concept of *end-to-end encryption (E2EE)* has been adopted to secure communications

doi:10.6342/NTU202401660

directly between users. The typical approach to achieving E2EE is to first establish a shared secret using public-key cryptography, and then encrypt messages using symmetric keys derived from this shared secret. For example, Signal protocol adopts Diffie-Hellman key agreement protocol and AES for message encryption [35].

The *Continuous Group Key Agreement (CGKA)* [3, 22] protocols are proposed to serve as the underlying key agreement protocols for secure group messaging. It is designed to *continuously* update keys to be resilient to key compromise events such as device breaches. These protocols ensure that keys generated outside the compromised periods remain confidential, thus protecting past messages from future compromises. Additionally, the asynchronous nature of CGKA protocols accommodates the reality that group members may not always be online simultaneously, providing necessary usability for practical use.

Several CGKA protocols have been proposed and undergone formal security analysis [1, 3, 5, 6, 22, 41]. Among these, the ART protocol [22] and its enhancement, TreeKEM [10], have garnered significant attention. These tree-based CGKA protocols utilize a binary tree structure known as a ratchet tree, which not only provides robust security properties but also maintains efficiency. Notably, TreeKEM has been incorporated into the ongoing IETF standard for secure group messaging, MLS (Message Layer Security) [9], underscoring its relevance and applicability in modern secure communication framework.

However, the security afforded by original CGKA protocols, such as ART and TreeKEM, is limited in its ability to provide user privacy. Notably, both protocols expose metadata like group membership and sender identities to external observers of network traffic. To

doi:10.6342/NTU202401660

mitigate these vulnerabilities, enhanced versions of these protocols have been proposed to hide such metadata [25, 29]. Moreover, modified ART have been proposed to conceal the sender identity from group members, thereby achieving sender anonymity within the group [17]. While the majority of research has concentrated on enhancing privacy against external adversaries, this work focuses on protecting user privacy against a specific type of group members, namely third-party applications or chatbots.

Chatbots, also known as third-party applications, are computer agents that can be added to group chats. They offer a range of functionalities, including bill splitting and event scheduling, and are supported by popular platforms such as Telegram, Discord, LINE, and Slack. The platforms allow chatbots to access user messages and metadata, just like a regular group member. However, this raises concerns regarding user privacy, as user messages may contain personal information, and chatbots are typically not trusted. Attackers can deploy malicious chatbots to spy on group activities, compromising the security of group messaging. Malicious chatbots can learn conversations and infer personal attributes like location and income using large language models (LLMs) [38], posing a serious threat to user privacy.

Various countermeasures have been implemented to address this privacy issue, though each has limitations. Telegram's Privacy Mode [39] limits chatbot access to only those messages that directly interact with them, significantly enhancing user privacy but not integrating with group E2EE. Similarly, Keybase uses a similar access control approach [32] and encrypts messages for chatbots with separate keys, which helps protect unrelated messages [31]. However, it does not provide strong E2EE properties, such as post-compromise security (PCS). Meanwhile, Slack's permission scope [37] allows workspace administrators to block chatbots from accessing user metadata, but lacks E2EE support.

doi:10.6342/NTU202401660

To our knowledge, no existing group messaging platform currently combines strong E2EE with effective protection against chatbots. This limitation stems from the absence of a CGKA protocol that incorporates both access control and metadata hiding capabilities. While there are existing CGKA protocols that support in-group sender anonymity [17], none yet offer access control. This gap motivates our research to develop a CGKA protocol that ensures user privacy against chatbots by integrating these crucial properties while maintaining robust E2EE.

Our approach is inspired by Balbas et al. [8], who developed an administrative CGKA (A-CGKA) protocol for securing group administration. We follow their methodology to extend CGKA protocols with additional security features. Our focus is on improving access control and metadata privacy, addressing security challenges for untrusted chatbots.

In this paper, we introduce an extended Continuous Group Key Agreement (CGKA) scheme, Our–CGKA, which uniquely incorporates chatbots as a specific type of group member and enables selective message access—features not available in previous CGKA schemes. We also present a new construction named CMRT, derived from TreeKEM and existing CGKA schemes. This construction modifies the traditional CGKA tree structure to support message access control and sender anonymity, while maintaining the security benefits of forward secrecy (FS) and post-compromise security (PCS). In summary, our contributions are as follows:

- We define the threat model and security properties for secure group messaging protocols that account for untrusted chatbots, as detailed in Chapter 3.
- We introduce Our–CGKA in Chapter 5.2, an extended CGKA protocol incorporating chatbot access control, with security definitions presented in Chapter 5.3.

doi:10.6342/NTU202401660

- We propose CMRT, a practical implementation of Our–CGKA in Chapter 6, demonstrating its adherence to security requirements and its computational efficiency in Chapter 7.

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

Chapter 2 Background

This chapter provides a brief background on secure messaging in group contexts and introduces Continuous Group Key Agreement (CGKA) protocols, the main focus of this work.

2.1 Secure Group Messaging

Modern messaging applications rely on service providers, the servers of messaging platforms, to buffer and deliver messages (as shown in Figure 2.1). This centralized design presents a risk, as a curious service provider could potentially eavesdrop on user messages. Therefore, the primary goal of secure messaging protocols is to ensure that only senders and receivers can decrypt messages using *end-to-end encryption (E2EE)*. E2EE guarantees that only the parties involved in the communication group can access the plaintext messages, and is now widely accepted as the security standard for messaging platforms. In addition, it is desirable that this security guarantee be resilient to key compromise.

To support E2EE, secure two-party messaging protocols were initially designed and subsequently extended to secure group messaging protocols. In the two-party secure messaging setting, Borisov et al. [14] considered potential key-compromise of secure messaging protocols, and showed that simply using public key cryptography, such as Pretty

doi:10.6342/NTU202401660

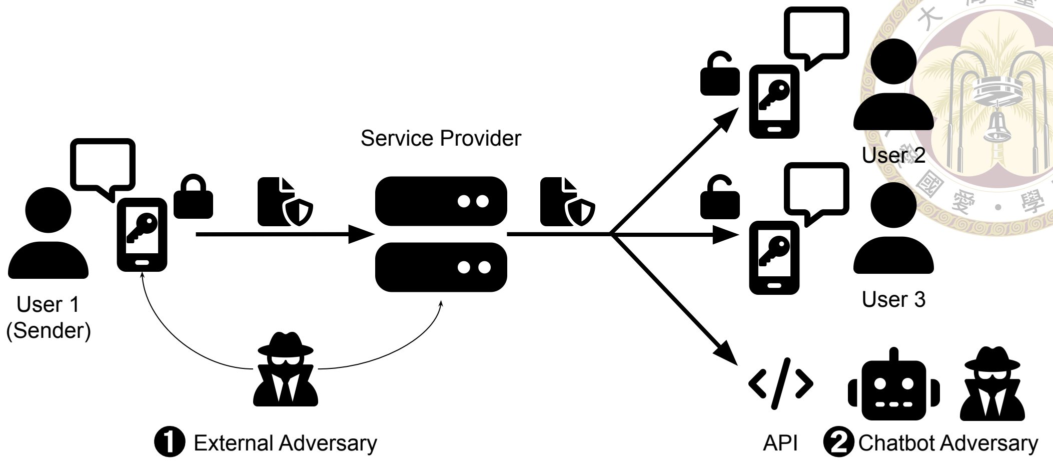


Figure 2.1: A typical messaging platform involves a service provider forwarding messages among group members. Two primary adversaries against users' privacy in this setting are ❶ malicious service providers with key-compromise capability and ❷ malicious chatbots. While state-of-the-art secure group messaging can address ❶ only, our work aims to address both.

Good Privacy (PGP), does not protect messages encrypted *before* the compromise. They propose Off-the-Record (OTR) protocol, where communicating parties continuously negotiate new Diffie-Hellman (DH) session keys and delete old session keys to achieve *forward secrecy (FS)* [28]. Cohn et al. [21] extend the concept of FS to protect message confidentiality and integrity *after* key compromise, known as *post-compromise security (PCS)*. They show that only stateful protocols can achieve PCS against full key compromise. Following the OTR protocol, the Double Ratchet Algorithm [35] adopts OTR's *ratcheting* design to create fresh session keys for each message exchanged between two parties. The Double Ratchet Algorithm serves as the underlying key agreement protocol for widely used messaging platforms such as WhatsApp, Signal, and Messenger's secret conversation, and is formally proven to satisfy both FS and PCS [2, 20].

To extend secure messaging from two to multiple parties, a straightforward design is to use pairwise secure channels between each two members, but the updating complexity is linear to the group size, lacking scalability for large groups. Another approach is

doi:10.6342/NTU202401660

the Sender Keys Protocol [42], which Signal and WhatsApp use for large groups; each member generates their own encryption key called a *sender key* and distributes the key to each group member through pairwise secure channels. The Sender Keys Protocol provides constant-time update and FS, but does not provide PCS [7, 11]. Lately, Continuous Group Key Agreement (CGKA) [3] becomes a unifying abstraction that captures support for both FS and PCS in group settings. CGKA has been instantiated by a number of group key protocols, such as Asynchronous Ratcheting Tree (ART) [22] and TreeKEM [10]. In particular, the variant of TreeKEM is the underlying group key protocol of Message Layer Security (MLS), an IETF standard [9] for secure group messaging. Although state-of-the-art secure group messaging, such as MLS, can defend against malicious service providers with key compromise capability (❶ in Figure 2.1), to our knowledge, no existing protocols can protect user privacy against malicious chatbots as well (❷ in Figure 2.1).

2.2 Continuous Group Key Agreement (CGKA)

Continuous Group Key Agreement (CGKA) [3, 22] are a class of group key agreement protocols designed to achieve asynchronous operations and strong security properties such as FS and PCS. A CGKA protocol supports three fundamental operations: key updating, adding a member, and removing a member, and continuously generates a group key shared by all current members. The resulting shared group keys should be confidential to those outside the group, achieving *key secrecy*, and should be frequently updated to maintain confidentiality from potential key compromise, achieving FS and PCS.

The tree-based CGKA protocols [33] leverage the tree structure to reduce update complexity to logarithmic. Specifically, Asynchronous Ratcheting Tree (ART) [22] is a

doi:10.6342/NTU202401660

tree-based CGKA protocol based on a Diffie-Hellman tree; TreeKEM [10], originated from ART, is based on a hash tree to enhance efficiency. The security of MLS, TreeKEM, and their variants have been thoroughly analyzed [3–5].

doi:10.6342/NTU202401660

Chapter 3 Problem Definition

This chapter identifies the adversaries in our threat model and outlines our assumptions about their capabilities and relationships. The proposed protocol aims to satisfy the desired security properties listed here in order to effectively defend against these adversaries.

3.1 Threat Model and Assumptions

We consider two types of adversaries: *malicious chatbots* and *external adversaries*, as shown in Figure 2.1. External adversaries include common adversaries considered in previous literature on secure messaging [34, 36, 40].

Chatbots We consider chatbots as insider adversaries who can participate in group conversations as regular members. The chatbots have access to the group messages, group metadata, and group events. In our scope, we assume that chatbots are passive adversaries. In other words, active attacks, such as sending malicious group modification messages [36], are out of scope. We assume that there is no collusion between a chatbot and a group member, or that the chatbot can trivially obtain all information that a member knows.

doi:10.6342/NTU202401660

External Adversaries We consider external adversaries who can observe network traffic and have key compromise capabilities, which allow an adversary to fully compromise a user's devices and then learn all states of a key agreement protocol [21]. It is assumed that a device may be compromised temporarily, but for only a finite period. Subsequently, it is expected that the user will regain control and execute at least one secure operation. A classic example is the service provider of a messaging service who has access to all encrypted user messages exchanged on the platform. We assume that the service provider is honest but curious, which means that while the service provider complies with the protocol, it may try to extract as much information as possible from the traffic it can access. We consider external adversaries as different adversaries from chatbot adversaries, and in this work we do not consider the case where external adversaries compromise chatbots or collude with each other. These cases are considered out of scope.

We assume that users are using an anonymous network like Tor [23], because without such tools to hide network-level metadata, such as source IP addresses, constructing a truly anonymous messaging protocol becomes infeasible.

3.2 Security Goals

With the two types of adversaries in mind, we identify security goals for a desired group key agreement protocol.

- **Forward Secrecy (FS)**: The shared keys generated *before* a compromise remains confidential to the external adversaries.

- **Post-Compromise Secrecy (PCS)**: The shared keys generated *after* a compromise

doi:10.6342/NTU202401660

remains confidential again to the external adversaries.

- **Sender Anonymity**: The chatbot adversary cannot effectively distinguish between the users in a group.

- **Selective Message Access**: The group keys should be confidential to the chatbot adversaries without access.

FS, and PCS are commonly required security properties for group key agreement protocols in the literature [3, 22, 36, 40]. Sender anonymity and selective message access are two additional security properties proposed to protect user privacy against chatbot adversaries.

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

Chapter 4 Notation

In group messaging protocols, there are two primary types of entities: users and chatbots. Users are distinguished by a unique user identifier, denoted as ID, while chatbots are identified by a chatbot identifier, CID.

In this paper, variable assignment is expressed as $a \leftarrow v$, indicating that variable a is set to value v. When a variable a is assigned the result of an algorithm Alg, it is denoted as $a \leftarrow \mathsf{Alg}(x)$ for deterministic outputs, and $a \leftarrow\$ \mathsf{Alg}(x)$ for randomized outputs. Uniform sampling from a set S is shown as $a \leftarrow\$ S$. A blank value is denoted by $\perp$. Storing and retrieving values in a dictionary D are represented by $D[k] \leftarrow v$ for storage, and $v \leftarrow D[k]$ for retrieval, respectively. To set all values in a dictionary to a single value v, the notation $D[\cdot] \leftarrow v$ is used.

In cryptographic protocols, the **assert** keyword enforces a condition within oracles and algorithms. If the condition is not met, execution halts and returns $\perp$. The keyword **public** in an oracle indicates that a variable is visible to adversaries; variables not marked as such are hidden from them.

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

Chapter 5 Continuous Group Key Agreement Protocol for Selective Message Access and Sender Anonymity

In this chapter, we first review Continuous Group Key Agreement (CGKA) by describing its syntax, then present our-CGKA, an extended scheme to model chatbots. Finally, we define the security of our-CGKA, which captures the security of CGKA, selective message access, and sender anonymity.

5.1 Formal Definition of Continuous Group Key Agreement

The Continuous Group Key Agreement (CGKA) protocols, described in [3, 5], provide a framework for secure communication between a group of users. These protocols ensure that all members of the group share a common secret, where the group can be dynamically adjusted to allow members to be added or removed. The shared secret can be periodically updated by any group member to maintain confidentiality of the secret. The

doi:10.6342/NTU202401660

syntax of the CGKA scheme is defined as follows.

Definition 1. *A continuous group key agreement (CGKA) scheme is defined as a tuple of the following algorithms* $\mathsf{CGKA} = (\mathsf{init}, \mathsf{create}, \mathsf{prop}, \mathsf{commit}, \mathsf{proc})$*:*

- $\gamma \leftarrow\$ \mathsf{init}(\mathsf{ID})$ *initializes the state* γ *for a user identified with* ID*.*
- $(\gamma', W) \leftarrow\$ \mathsf{create}(\gamma, \mathsf{ID}_1, \ldots, \mathsf{ID}_n)$ *creates a group with users* $\mathsf{ID_1}, \ldots, \mathsf{ID_n}$ *and outputs the updated state* γ' *and a control (welcome) message* W*.*
- $(\gamma', P) \leftarrow\$ \mathsf{prop}(\gamma, \mathsf{ID}, \mathsf{type})$ *creates a proposal for user* ID *with type* $\mathsf{type} \in \{\mathsf{add}, \mathsf{rem}, \mathsf{upd}\}$*, which corresponds to adding a user, removing a user, or updating the group secret. This outputs the updated state* γ' *and a proposal message* P*.*
- $(\gamma', T) \leftarrow\$ \mathsf{commit}(\gamma, \vec{P})$ *commits a list of proposals* $\vec{P}$ *and outputs the updated state* γ' *and a control message* T *for existing members or* W*.*
- $(\gamma', k) \leftarrow \mathsf{proc}(\gamma, T)$ *processes a control message* T *or* W *and outputs the updated state* γ' *and the new group key* k*.*

Design. The scheme allows group member to communicate through *control messages*, which can be generated by algorithms create or commit. The control messages contain essential information for other group members to synchronize the group information, including the group secret. The control messages are processed by the algorithm proc. Additionally, the scheme adopts a propose-and-commit style, allowing a user to consolidate multiple group operations into a single commit.

The intuition behind how the CGKA protocol achieves forward secrecy and post-compromise secrecy is that whenever a member joins, leaves, or performs an update, the

doi:10.6342/NTU202401660

group key is irreversibly updated. This update mechanism prevents new members from accessing previous keys and ensures that departing members cannot access future keys.

Protocol execution. To use the CGKA protocol, each user, identified by ID, begins by initializing their state using $\mathsf{init}(\mathsf{ID})$. A user can then create a group via create, providing a list of initial member identifiers. Users can *propose* to add members, remove them, or update her own key material. All of these operations will result in the update of group key. These proposals are aggregated and processed by the commit algorithm, which generates a control message T. Other group members synchronize their states by processing control messages of create or commit using the proc function, which produces a consistent group shared key k. The correctness of a CGKA protocol ensures that all group members obtain the same group key k from a commit message T.

Integration with Messaging Protocol. The CGKA scheme integrates seamlessly with instant messaging protocols by allowing control messages to be sent along with actual messages. For example, if Alice wants to remove Bob from a group and send a follow-up message, she sends a proposal rem to generate a control message T. After processing T to obtain the new group key k', Alice sends both T and her encrypted message using k' to the service provider, who then forwards them to the group members. Upon receipt, other group members process T to obtain k' and decrypt Alice's message using the new key.

5.2 Our CGKA

We extend the syntax of CGKA to include *chatbots* as a distinct type of group member, separate from *users*. In this extended framework, users retain unrestricted access to

doi:10.6342/NTU202401660

the shared keys, whereas chatbots are granted access only to specific keys. This restricted access is controlled by users who specify which chatbots can obtain a key update. They do this by providing a list of chatbot identifiers, $\mathsf{CID}_1, \ldots, \mathsf{CID}_n$, during the key update process invoked by upd. This differentiation facilitates selective message access, ensuring that each chatbot only accesses information for which it has explicit authorization.

Definition 2. *An Our-CGKA scheme is defined as a tuple of the following algorithms* $\mathsf{Our{-}CGKA} = (\mathsf{init}, \mathsf{create}, \mathsf{prop}, \mathsf{commit}, \mathsf{proc})$*:*

- *Algorithms* init, create, commit, proc *are identical as the* CGKA *scheme (Definition 1), where all operations can be invoked by both users and chatbots, except* create *which can only be invoked by users.*
- *In* prop*,* type *is extended to* $\mathsf{type} \in \{\mathsf{add}, \mathsf{rem}, \mathsf{upd}, \mathsf{add{-}cbt}, \mathsf{rem{-}cbt}, \mathsf{upd{-}cbt}\}$*, where* upd–cbt *can be used by chatbots only, and all the other types can be used by users only. The type* add–cbt *and* rem–cbt *can be used by users to add or remove a chatbot from group, and* upd–cbt *can be used by a chatbot to update the group secret.*
- $(\gamma', P) \leftarrow\$ \mathsf{prop}(\gamma, \mathsf{ID}, \vec{\mathsf{CID}}, \mathsf{type})$ *additionally takes a list of chatbot ids* $\vec{\mathsf{CID}}$*, which is only used for proposals of type* upd*. This allows a user to authorize a subset of chatbots to receive the new group secret, which is not possible in the original scheme.*

Protocol execution. Users create, manage groups, and update secrets in the same way as before. However, with the new functionality, users can now add or remove chatbots from the group using the add–cbt and rem–cbt proposals. The key shared with a chatbot

doi:10.6342/NTU202401660

member will only be updated if the corresponding chatbot identifier CID is specified in a upd proposal. In addition, chatbots can update their keys using upd–cbt proposals and handle control messages using the proc algorithm.

5.3 Security of Our CGKA

This section informally defines the security of our CGKA scheme, with the goal of clarifying our security goals in an accessible manner. A formal security definition is provided in Appendix A.2 for future formal security analysis.

The definitions for group key security, such as forward secrecy (FS), post-compromise secrecy (PCS), and selective message access, are based on the key indistinguishability game proposed by Alwen et al. [3]. The definition of sender anonymity is based on the sender indistinguishability game proposed by Chen et al. [17].

To define the security, we first define the epoch t, which is a protocol execution counter that advances whenever a control message is processed. Let γ_t denotes the state at epoch t and k_t denotes the group key at epoch t, we have the following relation for each group member: $(\gamma_t, k_t) \leftarrow \mathsf{proc}(\gamma_{t-1}, T)$ for any legitimate control message T.

Forward Secrecy (FS) For an external adversary who has access to all control messages T and compromises a member's state γ_t, the adversary should not be able to distinguish any key k_i for $i < t$ from a uniform random distribution.

Post-Compromise Secrecy (PCS) For an external adversary who has access to all control messages T and compromises a member's states, but a group member successfully

doi:10.6342/NTU202401660

creates a commit at epoch t without the adversary's control, i.e. the adversary knows nothing about this commit except the control message and the member's state before the commit, the adversary should not be able to distinguish any key k_i for $i > t$ from a uniform random distribution.

Sender Anonymity For a chatbot adversary within a group that has access to all control messages, the chatbot adversary should not be able to distinguish any control message T between any two group members with non-negligible probability.

Selective Message Access For a chatbot adversary within a group with access to all control messages, the chatbot adversary should not be able to distinguish any key k_i from a uniform random distribution if the commit corresponding to k_i excludes any update proposals that authorize that particular chatbot. In particular, there should be no upd proposal with the argument $\vec{\mathsf{CID}}$ containing the chatbot's identifier.

doi:10.6342/NTU202401660

Chapter 6 Compressed Multi-Roots Tree (CMRT)

In this chapter, we introduce the Compressed Multi-Roots Tree (CMRT) as a new construction for our Our–CGKA scheme. CMRT modifies the traditional tree structure of tree-based CGKA schemes to achieve sender anonymity and selective message access. The design leverages a single group shared secret to provide anonymity, dividing the structure into a "user subtree" and a "chatbot subtree" to hide user group details from chatbots. We further attempt to achieve selective message access by establishing parallel groups of users and each chatbot, allowing independent key updates. Although maintaining trees for each chatbot suggests a linear overhead, our design efficiently shares the user subtree across all trees. This creates a structure with multiple roots, which optimizes storage usage and scalability with respect to the number of chatbots.

The construction can be regarded as a wrapper protocol or an extension of an existing CGKA scheme. Although this work is based on the TreeKEM construction of a CGKA scheme, it can be substituted with another CGKA construction if the security of CGKA is maintained. We start by introducing the fundamental elements of our construction in Section 6.1, with a particular focus on TreeKEM. We then proceed to present an overview of our protocol in Section 6.2, followed by a more detailed description in Section 6.3.

doi:10.6342/NTU202401660

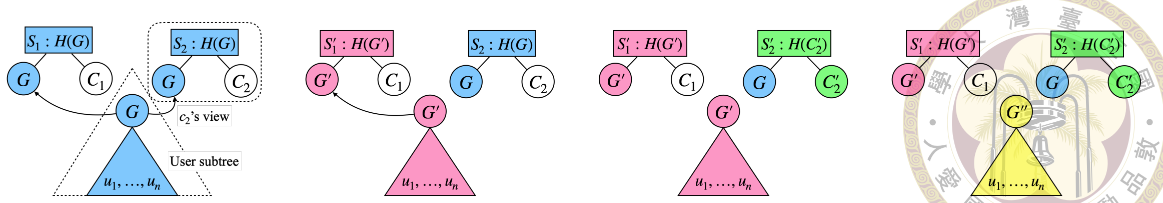
6.1 Building Blocks

Cryptographic Primitives The public-key encryption (PKE) scheme $\mathsf{PKE} = (\mathsf{PKEG}, \mathsf{PKEnc}, \mathsf{PKDec})$ consists of key generation $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{PKEG}(1^\lambda)$, encryption $c \leftarrow\!\!\$\ \mathsf{PKEnc}(\mathsf{pk}, m)$ using the public key pk, and decryption $m \leftarrow \mathsf{PKDec}(\mathsf{sk}, c)$ that retrieves the original message from the ciphertext. Additionally, a collision-resistant hash function $\mathsf{H} : \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$ is used. The formal definition for the security of these primitives is included in Appendix 9.

TreeKEM. TreeKEM [10] is a tree-based CGKA protocol constructed with a binary tree structure known as a *ratchet tree*. Each node in this tree holds a secret accessible only to the members within its subtree, and each member is assigned to a leaf node. The secret of the root node serves as the shared secret for the entire group. In TreeKEM, let $S_i \in \{0,1\}^\lambda$ denote a member's i-th secret from the leaf. The secret S_i is computed as the hash of the secret S_{i-1} from one of its child nodes, specifically the last child that updates the secret. Additionally, each node contains a pair of public-private keys (sk_i, pk_i) generated from its secret S_i using PKEG. The node information can be computed by $(sk_i, pk_i) \leftarrow \mathsf{PKEG}(S_i)$ where $S_i \leftarrow \mathsf{H}(S_{i-1})$.

To perform key update, a member (associated with one of the leaf nodes) randomly generates a new secret and iteratively computes the secrets along the path to the root node via hashing. The member then notifies other members of the new secrets by encrypting them with the public keys of the sibling nodes. Specifically, for a node v with a new

doi:10.6342/NTU202401660



(a) The users perform an update to obtain the group secret G and update the shared secrets for C_1, C_2. Secrets updated in this phase are colored blue.

(b) The users perform an update to obtain the group secret G' and update the shared secrets for C_1. Secrets updated in this phase are colored red.

(c) The chatbot C_2 performs an update. Secrets updated during this phase are colored green. Updates initiated by chatbots will not trigger an update for the user subtree.

(d) The users trigger an update only for user subtree, resulting the group secret G''. Secrets updated during this phase are colored yellow.

Figure 6.1: Illustration of CMRT with users $u_1, \ldots, u_n$ and chatbots c_1, c_2. Users share the group secret G from the *user subtrees* (triangles), while C_1, C_2 are secrets for chatbots c_1, c_2, respectively. The rectangles represent secrets shared between the group and each chatbot. The arrows indicate secret assignments, and the lines indicate parent-child relationships, where a child knows the secret of its parent. For example, in (b), G' is the group secret and S_1 is the secret shared between $u_1, \ldots, u_n$ and C_1.

secret S', the member encrypts S' with the public key of node $\mathsf{sibling}(v)$ and sends the ciphertext to the members under $\mathsf{sibling}(v)$, where $\mathsf{sibling}(v)$ denotes the sibling node of v. The member also publishes all new public keys along the updated path.

6.2 Protocol Overview

Before diving into the detailed description of our protocol, we provide an overview, initially discussing the two challenges achieving the new security properties and how our design addresses these issues.

Sender Anonymity. The primary reason that tree-based CGKA lacks sender anonymity is the exposure of sender identity through the update path, which is the path from the sender's node to the root. To address this, we modify the tree structure, where all group members are placed in the *user subtree*, while the *chatbot subtree* contains only the chatbot (as shown in Figure 6.1). During a user-initiated key update, the chatbot only needs to be aware of the root of the user subtree instead of the entire user subtree. This ensures that the chatbot can still compute a secret shared with all members without knowing the specific member initiating the update.

Selective Message Access. One main challenge in the secure messaging protocols is that some messages, along with the associated key updates, may be unavailable for chatbots. However, chatbots using traditional CGKA still require each key update message to maintain key consistency. To mitigate this, our construction maintains multiple root nodes, i.e., group keys, one per chatbot. Each chatbot shares its own state with the user group, and is updated only when a message is intended for that specific chatbot, as Figure 6.1 shows. If there is no key update for the chatbot, its root remains unchanged such that all group members and the chatbot continue to share the same secret.

The Compressed Multi-Roots Tree (CMRT) construction achieves the two additional security properties by allocating each chatbot a dedicated subtree while sharing a root node with the user subtree. From a user's perspective, the top of their tree connects to multiple root nodes, each linked to a chatbot. Conversely, each chatbot views itself in a smaller, 3-node tree connected only to the group members' subtree root. Key updates are managed efficiently, with users storing only necessary root nodes, thereby conserving space while ensuring asynchronous states across different chatbots, as shown in Figure 6.2. This "multi-root" structure is "compressed" to optimize data storage and maintain robust security simultaneously.

doi:10.6342/NTU202401660

6.3 Protocol

```
$\mathsf{init}(\mathsf{ID})$
$\gamma.s0 \leftarrow \mathsf{CGKA.init}(\mathsf{ID})$
$\gamma.\mathsf{ME} \leftarrow \mathsf{ID}$
$\gamma.\mathsf{cbts}[\cdot] \leftarrow \perp$
```

```
$\mathsf{create}(\mathsf{ID}_1, \dots, \mathsf{ID}_n)$
$(\gamma.s0, W_0) \leftarrow$
    $\mathsf{CGKA.create}(\mathsf{ID}_1, \dots, \mathsf{ID}_n)$
$\textbf{return } W_0$
```

```
$\mathsf{proc}(T)$
$(T_0, T_C) \leftarrow T$
$k \leftarrow \perp$
$\textbf{if } T_0 \neq \perp$    / from user
    $(\gamma.s0, k) \leftarrow \mathsf{proc}(\gamma.s0, T_0)$
    $(\mathsf{gsk}, \mathsf{gpk}) \leftarrow \mathsf{PKEG}(k)$
    $k' \leftarrow \mathsf{H}(k)$
    $k \leftarrow k'$
    $\textbf{for } (\mathsf{CID}, \cdot, \cdot) \in T_C$
        $\gamma.\mathsf{cbts}[\mathsf{CID}].\mathsf{gsk} \leftarrow \mathsf{gsk}$
    $\textbf{for } (\mathsf{CID}) \in T_C$
        $\gamma.\mathsf{cbts}[\mathsf{CID}] \leftarrow \perp$
$\textbf{else}$    / from chatbot
    $(\mathsf{CID}, \mathsf{cpk}, e) \leftarrow T_C$
    $(\mathsf{gsk}, \cdot) \leftarrow \mathsf{cbts}[\mathsf{CID}]$
    $k \leftarrow \mathsf{PKDec}_{\mathsf{gsk}}(e)$
    $\gamma.\mathsf{cbts}[\mathsf{CID}] \leftarrow (\mathsf{gsk}, \mathsf{cpk})$
$\textbf{return } k$
```

```
$\mathsf{prop}(\mathsf{ID}, \vec{\mathsf{CID}}, \mathsf{type})$
$\textbf{assert } \mathsf{type} \neq \mathsf{upd{-}cbt}$
$P \leftarrow \perp$
$\textbf{if } \mathsf{type} \in \{\mathsf{add{-}cbt}, \mathsf{rem{-}cbt}\}$
    $P \leftarrow (\mathsf{type}, \vec{\mathsf{CID}}_0)$
$\textbf{else}$
    $(\gamma.s0, P) \leftarrow \mathsf{CGKA.prop}(\mathsf{ID}, \mathsf{type})$
    $\textbf{if } \mathsf{type} = \mathsf{upd}\ P.\vec{\mathsf{CID}} \leftarrow \vec{\mathsf{CID}}$
$\textbf{return } P$
```

```
$\mathsf{commit}(\vec{P})$
$\vec{P} \leftarrow \mathsf{PropCleaner}(\vec{P})$
$(P_C, P_0) \leftarrow \mathsf{PropPartitioner}(\vec{P})$
$(\gamma.s0, T_0, k) \leftarrow \mathsf{commit}(\gamma.s0, P_0)$
$(\mathsf{gsk}, \mathsf{gpk}) \leftarrow \mathsf{PKEG}(k)$
$k' \leftarrow \mathsf{H}(k)$
$T_C \leftarrow \perp$
$\textbf{for } P \in P_C$
    $\textbf{if } P.\mathsf{type} = \mathsf{add{-}cbt}$
        $\mathsf{cpk} \leftarrow \mathsf{get{-}pk}(P.\mathsf{CID})$
        $\gamma.\mathsf{cbts}[P.\mathsf{CID}] \leftarrow (\mathsf{gsk}, \mathsf{cpk})$
        $T_C \leftarrow T_C \| (P.\mathsf{CID}, \mathsf{gpk}, \mathsf{PKEnc}_{\mathsf{cpk}}(k'))$
    $\textbf{if } P.\mathsf{type} = \mathsf{rem{-}cbt}$
        $\gamma.\mathsf{cbts}[P.\mathsf{CID}] \leftarrow \perp$
        $T_C \leftarrow T_C \| (\mathsf{CID})$
    $\textbf{if } P.\mathsf{type} = \mathsf{upd}$
        $\textbf{for } \mathsf{CID} \in P.\vec{\mathsf{CID}}$
            $(\cdot, \mathsf{cpk}) \leftarrow \gamma.\mathsf{cbts}[\mathsf{CID}]$
            $T_C \leftarrow T_C \| (\mathsf{CID}, \mathsf{gpk}, \mathsf{PKEnc}_{\mathsf{cpk}}(k'))$
            $\gamma.\mathsf{cbts}[\mathsf{CID}].\mathsf{gsk} \leftarrow \mathsf{gsk}$
$\textbf{return } (T_0, T_C)$
```

```
$\mathsf{init}(\mathsf{CID})$
$\gamma.\mathsf{ME} \leftarrow \mathsf{CID}$
$\gamma.\mathsf{gpk} \leftarrow \perp$
$k \leftarrow\!\!\$ \{0,1\}^\lambda$
$(\gamma.\mathsf{csk}, \gamma.\mathsf{cpk}) \leftarrow \mathsf{PKEG}(k)$
$\mathsf{set{-}pk}(\mathsf{CID}, \gamma.\mathsf{cpk})$
```

```
$\mathsf{prop}(\mathsf{ID}, \vec{\mathsf{CID}}, \mathsf{type})$
$\textbf{assert } \mathsf{type} = \mathsf{upd{-}cbt}$
$P \leftarrow (\mathsf{type}, \gamma.\mathsf{ME})$
$\textbf{return } P$
```

```
$\mathsf{commit}(\vec{P})$
$T_C \leftarrow \perp$
$\textbf{for } (\mathsf{type}, \mathsf{CID}) \in \vec{P} : \mathsf{type} = \mathsf{upd{-}cbt}$
        $\wedge\ \mathsf{CID} = \gamma.\mathsf{ME}$
    $k \leftarrow\!\!\$ \{0,1\}^\lambda$
    $(\gamma.\mathsf{csk}, \gamma.\mathsf{cpk}) \leftarrow \mathsf{PKEG}(k)$
    $k' \leftarrow \mathsf{H}(k)$
    $\mathsf{gpk} \leftarrow \gamma.\mathsf{gpk}$
    $T_C \leftarrow (\gamma.\mathsf{ME}, \gamma.\mathsf{cpk}, \mathsf{PKEnc}_{\mathsf{gpk}}(k'))$
$\textbf{return } ((\perp, T_C), k')$
```

```
$\mathsf{proc}(T)$
$(\cdot, T_C) \leftarrow T$
$(\cdot, \gamma.\mathsf{gpk}, e) \leftarrow T_C$
$\mathsf{csk} \leftarrow \gamma.\mathsf{csk}$
$k \leftarrow \mathsf{PKDec}_{\mathsf{csk}}(e)$
$\textbf{return } k$
```

Figure 6.2: The CMRT protocol. Unboxed algorithms are those called by the *users* (i.e. $\gamma.\mathsf{ME} \in \mathsf{ID}$), while $\boxed{\text{boxed}}$ algorithms are those called by the *chatbots*. We assume that the users will only call the unboxed algorithms and the chatbots will only call the $\boxed{\text{boxed}}$ algorithms.

This section presents a detailed description of the construction. The construction is divided into two parts: algorithms that can only be used by users (unboxed algorithms in Figure 6.3) and algorithms that can only be used by chatbots ($\boxed{\text{boxed}}$ algorithms in Figure 6.3). For simplicity, this work only considers a single group setting, where each user and chatbot is limited to joining one group. However, the protocol can be extended to support multiple groups by using group identifiers to distinguish between them. Please

doi:10.6342/NTU202401660

note that the security of multiple group settings is not covered in this work.

Our construction is based on an underlying CGKA protocol. To differentiate related variables, those directly associated with the CGKA protocol are subscripted with “0.” For example, variables such as s_0, W_0, and T_0 are marked in this way to distinguish them from other elements within the scheme.

We will first introduce the PKI model used in our construction, then explain how to initialize states using init, how to create the group using create, how to create a proposal using prop, how to commit proposals using commit, and how to process a commit message using proc.

PKI. We assume the presence of an incorruptable PKI, typically managed by the service provider, to handle the public keys of chatbots. During the initialization phase, a chatbot identified by CID generates a new key pair $(\mathsf{csk}, \mathsf{cpk})$, registering the public key cpk with the PKI using $\mathsf{set–pk}(\mathsf{CID}, \mathsf{cpk})$. Users can subsequently retrieve this public key using $\mathsf{get–pk}(\mathsf{CID})$.

Initialization of states. The states for each entity, including a user or a chatbot, are denoted as γ. A user’s state records the state of the underlying CGKA protocol as $\gamma.s0$, its identifier $\gamma.\mathsf{ME}$, and a dictionary $\gamma.\mathsf{cbts}$ indexed by chatbot identifier to record information for each chatbot in a group. A chatbot’s state records its identifier $\gamma.\mathsf{ME}$, the user subgroup’s public key $\gamma.\mathsf{gpk}$, and a PKE key pair $(\gamma.\mathsf{csk}, \gamma.\mathsf{cpk})$ for its single chatbot subgroup.

doi:10.6342/NTU202401660

Group creation. Only a user can create a group, which initially contains users only. Our construction delegates the request to the underlying CGKA protocol, and stores the state of the user group as $\gamma.s0$, and returns the welcome message W_0.

Proposals. In our scheme, users are permitted to create various types of proposals, with the exception of the upd–cbt type. For the add–cbt and rem–cbt types, the proposal created by CMRT includes the proposal type and the first element of vector $\vec{\mathsf{CID}}$, which serves to identify the chatbot to be added or removed. For other types, such as add, rem, and upd, the request is handled by the underlying CGKA protocol. In particular, for proposals of type upd, the vector $\vec{\mathsf{CID}}$ is also included to specify the chatbots authorized to be updated. Conversely, a chatbot can only initiate a proposal of type upd–cbt to request a key update.

Commits. The commit algorithm for users in the CGKA protocol incorporates several steps: (1) Using PropCleaner, it ensures that $\vec{P}$ is a valid proposal sequence by removing duplicate proposals and prioritizing removal proposals. (2) Using PropPartitioner, it divides the proposals into ordinary CGKA proposals (P_0) and chatbot-related proposals (P_C). (3) It applies P_0 within the user subgroup using CGKA.commit, generating a control message T_0 and a new key k. (4) It computes a PKE keypair $(\mathsf{gsk}, \mathsf{gpk})$ for the user subgroup and a root key k'. (5) Chatbot-related proposals are processed individually, involving steps for adding, removing, or updating chatbots, each involving specific actions managing public/private keys and appending encrypted root key to the chatbot control message T_C. (6) Finally, the algorithm outputs control messages (T_0, T_C) and the new shared group key k'.

The commit algorithm for a chatbot handles upd–cbt proposals by following these

doi:10.6342/NTU202401660

steps: (1) It validates each proposal. (2) It randomly generates a new chatbot secret k. (3) From k, it computes the PKE key pair $(\mathsf{csk}, \mathsf{cpk})$ and the new root key k'. (4) It retrieves the user group's public key gpk from the state γ. (5) It appends a control message including the chatbot identifier, the chatbot public key cpk, and the encrypted root key k' under gpk.

Processing control messages. The proc algorithm for users processes control messages in two scenarios: (1) If the control message is from another user, it delegates the ordinary control message T_0 to the underlying CGKA protocol, computes the new PKE key pair and new root key, and updates chatbot states based on the control messages received. (2) If the control message is from a chatbot, it decrypts the root key using the previously stored group secret key gsk. The state of the underlying CGKA protocol remains unchanged in this case.

The proc algorithm for chatbots focuses solely on handling control messages that pertain to chatbots. It updates the group public key gpk and decrypts the new root key using the chatbot's previously stored private key.

doi:10.6342/NTU202401660

Chapter 7 Results

7.1 Security Analysis

This section provides a security analysis to show that CMRT satisfies the desired security properties. The analysis is based on the assumption that the underlying CGKA scheme is secure, i.e. it satisfies forward secrecy and post-compromise secrecy. We use this fact to conclude that CMRT also satisfies two two properties, while also achieving selective message access. For sender anonymity, the analysis relies on the assumption of a membership-hiding CGKA scheme, which prevents external adversaries from learning any information about the sender's identity.

In our analysis, we assume TreeKEM as the underlying CGKA scheme, since it has been shown to satisfy forward secrecy and post-compromise secrecy [3]. Furthermore, the method proposed by Emura et al. [25] allows us to adapt TreeKEM into a membership-hiding CGKA that satisfies our security requirements.

We assume that the hash function H, used in both TreeKEM and CMRT, functions as a pseudorandom generator (PRG). This implies that if H receives uniformly random input, its output will also be uniformly random. Furthermore, we assume the public key encryption (PKE) scheme employed is IND-CPA secure, which guarantees that an adver-

doi:10.6342/NTU202401660

sary cannot distinguish between the ciphertexts of any two chosen plaintexts. The formal security definition for these cryptographic primitives is included in Appendix 9.

7.1.1 Forward Secrecy

Assume that no chatbot is compromised, we show that for an external adversary with messages T and compromises a member's state γ_t, any key k_i for $i < t$ is indistinguishable from uniform random distribution.

First, the adversary gains no information from the compromised keys. The underlying CGKA protocol ensures forward secrecy by guaranteeing that any key k generated prior to compromise is indistinguishable from a uniform random distribution. Furthermore, by definition of a pseudorandom generator (PRG), the group keys k' generated by CMRT before the compromise also retain this indistinguishability, since each k' is derived via $k' = \mathsf{PRG}(k)$.

Second, the adversary gains no information from the control messages, which contain past keys encrypted under the chatbots' public keys. Since the PKE scheme is IND-CPA secure, the adversary cannot distinguish between the keys and uniform random values based on the ciphertext alone.

7.1.2 Post-Compromise Secrecy

Assume that no chatbot is compromised, we show that for an external adversary with messages T and compromises a member's states, yet a group member successfully commits an uncompromised operation at epoch t, the key k_i for $i > t$ is indistinguishable from uniform random distribution.

doi:10.6342/NTU202401660

This claim can be supported by a similar argument used to demonstrate forward secrecy. Namely, the group keys generated after a compromise are indistinguishable from a uniform random distribution according to the properties of PRG. In addition, the IND-CPA security of PKE scheme used prevents an attacker from obtaining any information from the control messages.

7.1.3 Sender Anonymity

Assume that the underlying CGKA scheme is metadata-hiding, and the new leaf secret chosen during commit is uniformly random, we show that the chatbot adversary is unable to distinguish any control message T between any two group members with non-negligible probability.

First, by definition, the adversary cannot distinguish the control messages from the metadata-hiding CGKA scheme. Second, the adversary is unable to distinguish the additional control messages generated by CMRT, since the control message only contains chatbot identifier, group public key, and a ciphertext. The ciphertext indistinguishable according to IND-CPA security. Furthermore, the chatbot identifier remains constant across different members, offering no distinct information. The group public key, derived from the root secret which itself is produced through a series of PRG operations starting from a leaf secret assumed to be sampled from a uniform random distribution, also maintains indistinguishability. The group key itself is also indistinguishable, according to the property of PRG.

doi:10.6342/NTU202401660

7.1.4 Selective Message Access

We show that a chatbot adversary within a group is unable to distinguish any unauthorized key k from a uniform random distribution.

This can be demonstrated by examining its compliance with both forward secrecy and post-compromise secrecy. Consider a chatbot adversary $\mathcal{A}$ that obtains authorization with a key k_t at epoch t. Under forward secrecy, all keys k_i for $i < t$ remain secure and indistinguishable from a uniform random distribution because they are generated via a PRG from a uniformly random source. Furthermore, since the control message does not contain any ciphertext encrypted with the public key of $\mathcal{A}$, it maintains its indistinguishability according to IND-CPA security. Conversely, post-compromise secrecy ensures that all keys k_i for $i > t$ are indistinguishable from $\mathcal{A}$, thus protecting subsequent keys and ensuring that the chatbot adversary does not learn any information about the unauthorized keys between the authorized keys.

7.2 Security Comparison

	E2EE		Sender Anonymity	SMA
	FS	PCS		
Sender Keys	✓	✗	✗	✗
Keybase	✓	✗	✗	✓
TreeKEM, ART	✓	✓	✗	✗
AART	✓	✓	✓	✗
CMRT	✓	✓	✓	✓

Table 7.1: Security comparisons between group key agreement protocols. SMA stands for Selective Message Access.

Table 7.1 presents a comparative analysis of the security features provided by widely used key agreement protocols in secure messaging systems. The Sender Keys proto-

doi:10.6342/NTU202401660

col [42] provides Forward Secrecy (FS) but does not support Post-Compromise Security (PCS). Keybase's key derivation algorithm provides optional forward secrecy and dedicated decryption keys for bots to achieve selective message access, but it doesn't provide PCS [30, 31]. CGKA protocols, such as TreeKEM [10] and ART [22], provide both FS and PCS, but lack privacy features, such as sender anonymity and selective message access. AART [17] improves on ART by incorporating sender anonymity, but still falls short in providing selective message access. Our protocol is the first to support all the aforementioned properties.

7.3 Efficiency Analysis

		# public-key operations			# symmetric operations		
		sender	per user	per chatbot	sender	per user	per chatbot
Sender Keys	setup	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$
	ongoing	0	0	0	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
ART, AART	setup	$\mathcal{O}(n+m)$	$\mathcal{O}(\log(n+m))$	$\mathcal{O}(\log(n+m))$	0	0	0
	ongoing	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
TreeKEM	setup	$\mathcal{O}(n+m)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n+m)$	$\mathcal{O}(\log(n+m))$	$\mathcal{O}(\log(n+m))$
	ongoing	$\mathcal{O}(\log(n+m))$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n+m))$	$\mathcal{O}(\log(n+m))$	$\mathcal{O}(\log(n+m))$
CMRT (Ours)	setup	$\mathcal{O}(n+m)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n+m)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
	ongoing	$\mathcal{O}(\log n+m)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n+m)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$

Table 7.2: Computation complexity comparison. n = number of group members, m = number of chatbots.

This section analyzes the computational overhead of CMRT and compares it to other secure group messaging protocols. Table 7.2 shows the computational complexity of our protocol. The *setup phase* involves creating a group of n users and m chatbots, and the *ongoing phase* involves both adding a chatbot to the group and sending a message to the chatbot.

Setup phase. For setup phase, the group initiator uses $\mathcal{O}(n+m)$ PKE operations and $\mathcal{O}(n+m)$ symmetric operations. The construction of the TreeKEM with n members involves $\mathcal{O}(n)$ PKE operations and hash operations, respectively. To compute the shared

doi:10.6342/NTU202401660

secret for each chatbot, the initiator also performs $\mathcal{O}(m)$ PKE operations and hash operations, respectively. For the receivers, each user requires $\mathcal{O}(1)$ PKE operations and $\mathcal{O}(\log n)$ hash operations to initiate the TreeKEM. For each chatbot, it takes $\mathcal{O}(1)$ PKE operations to decrypt the secret, but only $\mathcal{O}(1)$ hash operations to compute the shared secret due to the unbalanced tree structure.

Ongoing phase. Suppose a user sends a message to the chatbots. The message sender performs $\mathcal{O}(\log n + m)$ PKE operations and symmetric operations, respectively. Updating the TreeKEM involves $\mathcal{O}(\log n)$ public key generations and hash operations. Each chatbot takes $\mathcal{O}(1)$ PKE operations and hash operations for the sender to do the key update and message encryption, respectively, and there are m chatbots, imposing $\mathcal{O}(m)$ overhead. Message recipients, including users and chatbots, perform identical actions as in the setup phase to update the secret, resulting in the same overhead.

Adding a chatbot to the group is almost the same as sending a message to a chatbot. The initiator performs $\mathcal{O}(\log n)$ PKE operations and symmetric operations, respectively, to update the TreeKEM. Both the chatbot and other members perform the same actions as in the setup phase to update the secret.

doi:10.6342/NTU202401660

Chapter 8 Related Work

This chapter reviews the existing literature related to our work. First, we explore the security and privacy issues associated with chatbots, including existing countermeasures in Section 8.1. Second, we examine secure group messaging protocols aimed at hiding metadata in Section 8.2.

8.1 Chatbot Security

Several studies have conducted large-scale security evaluations of chatbots on modern messaging platforms. Edu et al. [24] analyzed over 15,000 Discord chatbots and found that over 40% of the chatbots examined ask for permission to access message history, but less than 5% of them offer a privacy policy. This lack of transparency raises concerns regarding how developers store and utilize the collected user activity. Similarly, Chen et al. [19] analyzed design flaws in chatbot-like third-party apps on Business Collaboration Platforms (BCP), such as Slack. Their analysis showed that these apps can steal messages or impersonate users. These studies underscore the privacy risks associated with chatbots, providing strong motivation for our work.

In addition to empirical security evaluation, some related work focus on developing solutions to protect users' privacy from chatbots. Biswas [12] highlighted the potential for

doi:10.6342/NTU202401660

users to inadvertently disclose sensitive information such as location to chatbots. Their solution combines Named Entity Recognition (NER) and Public Key Encryption with Keyword Search (PEKS) [13] to achieve server-aided access control. Their solution allows the service provider to filter messages based on the encrypted keywords provided by the chatbots, without knowing the keywords and messages. While their objective aligns with our aim of selective message access, their solution results in high computation overhead linear to the number of keywords, and requires modification to be compatible with group settings and satisfy FS and PCS.

Some modern messaging platforms have implemented access controls for chatbots. Telegram's Privacy Mode [39] restricts chatbots to only accessing messages that directly mention them, contain their predefined commands, or are replies to such messages. Slack implements fine-grained permission controls [37] for accessing user metadata. However, neither platform combines these access controls with group end-to-end encryption (E2EE). Keybase [31] offers a similar policy to Telegram's and uses dedicated keys for encrypting messages accessible to chatbots, but it lacks a mechanism to update keys for post-compromise security (PCS).

8.2 Metadata-hiding Secure Messaging

Several research aims at extending secure group messaging to hide metadata. Chen et al. [17, 18] formalized the notion of Internal/External Group Anonymity (IGA/EGA), preserving sender indistinguishability from the perspective of group insiders and outsiders. Anonymous Asynchronous Ratchet Tree (AART) is proposed to achieve sender anonymity, FS, and, PCS. Our work adopts a definition similar to IGA that accommo-

doi:10.6342/NTU202401660

dates the presence of chatbots.

Several work focus on hiding metadata from outsiders of the group, including service providers. Signal's Private Group hides the group membership from the service providers using key-verified anonymous credentials [15]. Keita et al. [25] define three layers of metadata from the perspective of group outsiders, and propose wrapper protocols that upgrade existing non-metadata-hiding CGKA to metadata-hiding ones. Their design utilizes the fact that the single group shared key provides anonymity, and the shared key is used to encrypt While our work can also be seen as a wrapper protocol, we focus on threats by group insiders (in our case, chatbots) and modifying CGKA to protect users' privacy from chatbots.

Keita et al. [26, 27] proposed a scheme to achieve both E2EE and anonymous authentication within a group, where a sender chooses an arbitrary ID as an ephemeral identity, which is signed using the group signature [16] and sent to a receiver to prove membership anonymously.

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

Chapter 9 Conclusions and Future Directions

In this work, we address the absence of a group messaging protocol that simultaneously supports strong E2EE and chatbot isolation. We propose a security model for secure group messaging with chatbots, including an extended CGKA scheme, designated as Our–CGKA, that supports message access control. Additionally, and security definition capturing strong E2EE properties and desired properties for chatbot isolation: sender anonymity, and selective message access. We introduce CMRT as an construction for Our–CGKA, which manages multiple keys among group members and chatbots while maintaining sender anonymity and access control. Through security analysis and theoretical analysis, we demonstrated that CMRT achieves these properties effectively without imposing significant overhead.

This work initiates a critical discussion on the development of secure messaging protocols that include untrusted entities such as chatbots. It highlights the need for future research to explore the practicality of these designs and to identify opportunities for enhancing performance. A formal security analysis is also critical for strong security guarantee. This exploration is essential for advancing the security of daily communications.

doi:10.6342/NTU202401660

doi:10.6342/NTU202401660

References

[1] J. Alwen, B. Auerbach, M. C. Noval, K. Klein, G. Pascual-Perez, K. Pietrzak, and M. Walter. Cocoa: concurrent continuous group key agreement. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 815–844. Springer, 2022.

[2] J. Alwen, S. Coretti, and Y. Dodis. The double ratchet: security notions, proofs, and modularization for the signal protocol. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 129–158. Springer, 2019.

[3] J. Alwen, S. Coretti, Y. Dodis, and Y. Tselekounis. Security analysis and improvements for the ietf mls standard for group messaging. In Annual International Cryptology Conference, pages 248–277. Springer, 2020.

[4] J. Alwen, S. Coretti, Y. Dodis, and Y. Tselekounis. Modular design of secure group messaging protocols and the security of mls. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pages 1463–1483, 2021.

[5] J. Alwen, S. Coretti, D. Jost, and M. Mularczyk. Continuous group key agreement with active security. In Theory of Cryptography, pages 261–290. Springer, 2020.

doi:10.6342/NTU202401660

[6] J. Alwen, M. Mularczyk, and Y. Tselekounis. Fork-resilient continuous group key agreement. In Annual International Cryptology Conference, pages 396–429. Springer, 2023.

[7] D. Balbás, D. Collins, and P. Gajland. Analysis and improvements of the sender keys protocol for group messaging. In XVII Reunión española sobre criptología y seguridad de la información. RECSI 2022, volume 265, page 25. Ed. Universidad de Cantabria, 2022.

[8] D. Balbás, D. Collins, and S. Vaudenay. Cryptographic administration for secure group messaging. In 32nd USENIX Security Symposium (USENIX Security 23), pages 1253–1270, 2023.

[9] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. RFC 9420, July 2023.

[10] K. Bhargavan, R. Barnes, and E. Rescorla. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS). Research report, Inria Paris, May 2018.

[11] A. Bienstock, J. Fairoze, S. Garg, P. Mukherjee, and S. Raghuraman. A more complete analysis of the signal double ratchet algorithm. In Annual International Cryptology Conference, pages 784–813. Springer, 2022.

[12] D. Biswas. Privacy preserving chatbot conversations. In IEEE International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 2020.

[13] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In Advances in Cryptology - EUROCRYPT 2004, pages 506–522. Springer, 2004.

doi:10.6342/NTU202401660

[14] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use pgp. In Proceedings of the 2004 ACM workshop on Privacy in the electronic society, pages 77–84, 2004.

[15] M. Chase, T. Perrin, and G. Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 1445–1459, 2020.

[16] D. Chaum and E. Van Heyst. Group signatures. In Advances in Cryptology—EUROCRYPT' 91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10, pages 257–265. Springer, 1991.

[17] K. Chen and J. Chen. Anonymous end to end encryption group messaging protocol based on asynchronous ratchet tree. In Information and Communications Security, pages 588–605. Springer, 2020.

[18] K. Chen, J. Chen, and J. Zhang. Anonymous asynchronous ratchet tree protocol for group messaging. Sensors, 21(4):1058, 2021.

[19] Y. Chen, Y. Gao, N. Ceccio, R. Chatterjee, K. Fawaz, and E. Fernandes. Experimental security analysis of the app model in business collaboration platforms. In 31st USENIX Security Symposium (USENIX Security 22), pages 2011–2028, 2022.

[20] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the signal messaging protocol. Journal of Cryptology, 33:1914–1983, 2020.

doi:10.6342/NTU202401660

[21] K. Cohn-Gordon, C. Cremers, and L. Garratt. On post-compromise security. In 2016 IEEE 29th Computer Security Foundations Symposium (CSF), 2016.

[22] K. Cohn-Gordon, C. Cremers, L. Garratt, J. Millican, and K. Milner. 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, pages 1802–1819, 2018.

[23] R. Dingledine, N. Mathewson, P. F. Syverson, et al. Tor: The second-generation onion router. In USENIX security symposium, volume 4, pages 303–320, 2004.

[24] J. Edu, C. Mulligan, F. Pierazzi, J. Polakis, G. Suarez-Tangil, and J. Such. Exploring the security and privacy risks of chatbots in messaging services. In Proceedings of the 22nd ACM internet measurement conference, 2022.

[25] K. Emura, K. Kajita, R. Nojima, K. Ogawa, and G. Ohtake. Membership privacy for asynchronous group messaging. In International Conference on Information Security Applications, pages 131–142. Springer, 2022.

[26] K. Emura, A. Kanaoka, S. Ohta, K. Omote, and T. Takahashi. Secure and anonymous communication technique: Formal model and its prototype implementation. IEEE Transactions on Emerging Topics in Computing, 4(1):88–101, 2015.

[27] K. Emura, A. Kanaoka, S. Ohta, and T. Takahashi. Building secure and anonymous communication channel: Formal model and its prototype implementation. In Proceedings of the 29th Annual ACM Symposium on Applied Computing, pages 1641–1648, 2014.

[28] C. G. Günther. An identity-based key-exchange protocol. In Advances in Cryptology — EUROCRYPT '89, pages 29–37. Springer Berlin Heidelberg, 1990.

doi:10.6342/NTU202401660

[29] K. Hashimoto, S. Katsumata, and T. Prest. How to hide metadata in mls-like secure group messaging: simple, modular, and post-quantum. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pages 1399–1412, 2022.

[30] Keybase. Chat - crypto | keybase docs. https://book.keybase.io/docs/chat/ephemeral. Accessed on 2024-06-10.

[31] Keybase. Chat - restricted bots | keybase docs. https://keybase.io/docs/chat/restricted_bots. Accessed on 2024-07-10.

[32] Keybase. Introducing keybase bots. https://keybase.io/blog/bots. Accessed on 2024-06-10.

[33] Y. Kim, A. Perrig, and G. Tsudik. Tree-based group key agreement. ACM Transactions on Information and System Security (TISSEC), 7(1):60–96, 2004.

[34] M. Marlinspike and T. Perrin. The x3dh key agreement protocol. https://www.signal.org/docs/specifications/x3dh/, 2016.

[35] T. Perrin and M. Marlinspike. The double ratchet algorithm. https://signal.org/docs/specifications/doubleratchet/, Nov 2016. Accessed on 2024-02-07.

[36] P. Rösler, C. Mainka, and J. Schwenk. More is less: on the end-to-end security of group chats in signal, whatsapp, and threema. In 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pages 415–429. IEEE, 2018.

[37] Slack. Permission scopes. https://api.slack.com/scopes. Accessed on 2024-01-29.

doi:10.6342/NTU202401660

[38] R. Staab, M. Vero, M. Balunović, and M. Vechev. Beyond memorization: Violating privacy via inference with large language models. arXiv preprint arXiv:2310.07298, 2023.

[39] Telegram Messenger Inc. Telegram privacy policy. https://telegram.org/privacy, Apr 2023. Accessed on 2024-02-07.

[40] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. Sok: Secure messaging. In 2015 IEEE Symposium on Security and Privacy, pages 232–249, 2015.

[41] M. Weidner, M. Kleppmann, D. Hugenroth, and A. R. Beresford. Key agreement for decentralized secure group messaging with strong security guarantees. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pages 2024–2045, 2021.

[42] WhatsApp. Whatsapp encryption overview. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf, Jan 2023. Accessed on 2024-02-07.

doi:10.6342/NTU202401660

Appendix A — Cryptographic Primitives

A.1 Pseudorandom Generators

A pseudorandom generator (PRG) is a function $\mathsf{PRG} : \mathcal{W} \to \mathcal{W}$ that produces output $\mathsf{PRG}(U)$, which is indistinguishable from a uniformly random $U' \in \mathcal{W}$, given U also uniformly sampled from $\mathcal{W}$. The security of a PRG is measured by the advantage for an attacker $\mathcal{A}$ has in distinguishing between $\mathsf{PRG}(U)$ and U', denoted as $\mathsf{Adv}_{prg}^{\mathsf{PRG}}(\mathcal{A})$.

Definition 3. *A* PRG *scheme is* (t, ϵ)*-CPA-secure if for all adversary* $\mathcal{A}$ *with running time* t*,*

$$\mathsf{Adv}_{prg}^{\mathsf{PRG}}(\mathcal{A}) \leq \epsilon$$

A.2 Public Key Encryption

A public-key encryption (PKE) scheme $\mathsf{PKE} = (\mathsf{PKEG}, \mathsf{PKEnc}, \mathsf{PKDec})$ consists of the following algorithms:

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow\$ \mathsf{PKEG}(s)$: Generates a PKE key pair from a secret s.

doi:10.6342/NTU202401660

- $c \leftarrow\$ \mathsf{PKEnc}(\mathsf{pk}, m)$: Encrypts the message m using the public key pk and outputs a ciphertext c.
- $m \leftarrow \mathsf{PKDec}(\mathsf{sk}, c)$: Decrypts the ciphertext c using sk and outputs the message m.

IND-CPA security. Consider the following game:

```
IND-CPA$^{\mathcal{A}}_{\mathsf{PKE}}(\lambda)$
$b \leftarrow\$ \{0,1\}$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow\$ \mathsf{PKEG}(1^\lambda)$
$(m_0, m_1) \leftarrow\$ \mathcal{A}(1^\lambda, \mathsf{pk})$
$c \leftarrow\$ \mathsf{Enc}(\mathsf{pk}, m_b)$
$b' \leftarrow\$ \mathcal{A}(1^\lambda, c)$
return $1_{b=b'}$
```

Definition 4. *A* PKE *scheme is (t, ϵ)-CPA-secure if for all adversary $\mathcal{A}$ with running time t,*

$$\mathsf{Adv}^{\mathsf{PKE}}_{\text{IND-CPA}}(\mathcal{A}) \le \epsilon$$

doi:10.6342/NTU202401660

Appendix B — Formal Security Definition for Our CGKA

This appendix chapter outlines the formal security definitions for Our–CGKA, which comprises two parts: (1) standard CGKA security properties including key secrecy, forward secrecy (FS), and post-compromise secrecy (PCS), and (2) new security properties specific to our protocol, namely sender anonymity and selective message access.

```
$\mathcal{O}^{\mathsf{Setup}}()$
1 :  $\mathsf{K}[\cdot], \mathsf{ST}[\cdot] \leftarrow \perp$
2 :  public $\mathsf{T}[\cdot], \mathsf{usrs}[\cdot], \mathsf{cbts}[\cdot] \leftarrow \perp$
3 :  $\mathsf{ep}[\cdot], \mathsf{C}[\cdot] \leftarrow 0$
4 :  $\mathsf{prop\text{–}ctr}, \mathsf{comm\text{–}ctr}, \mathsf{exp\text{–}ctr} \leftarrow 0$
5 :  $\mathsf{ST}[\mathsf{ID}] \leftarrow \mathsf{init}(1^\lambda, \mathsf{ID}) \forall \mathsf{ID}$
```

```
$\mathcal{O}^{\mathsf{Prop}}(\mathsf{ID}, \mathsf{ID}', \vec{\mathsf{CID}}, \mathsf{type})$
1 :  $(\gamma, P) \leftarrow\$ \mathsf{prop}(\mathsf{ST}[\mathsf{ID}], \mathsf{ID}', \vec{\mathsf{CID}}, \mathsf{type})$
2 :  $\mathsf{T}[\mathsf{ep}[\mathsf{ID}], \text{‘}prop', \text{++}\mathsf{prop\text{–}ctr}] \leftarrow P$
3 :  $\mathsf{ST}[\mathsf{ID}] \leftarrow \gamma$
```

```
$\mathcal{O}^{\mathsf{Create}}(\mathsf{ID}_0, \mathsf{ID}_1, \ldots, \mathsf{ID}_n)$
1 :  $(\gamma, T) \leftarrow \mathsf{create}(\mathsf{ST}[\mathsf{ID}_0], \mathsf{ID}_1, \ldots, \mathsf{ID}_n)$
2 :  if $T = \perp$ return
3 :  $\mathsf{T}[t, \mathsf{ID}_0, \mathsf{ID}_i] \leftarrow T \forall$
4 :  $\mathsf{ST}[\mathsf{ID}_0] = T$
```

```
$\mathcal{O}^{\mathsf{Commit}}(\mathsf{ID}, (i_1, \ldots, i_k))$
1 :  $\vec{P} \leftarrow\$ (T[\mathsf{ID}, \text{‘}prop', i])_{i=(i_1,\ldots,i_k)}$
2 :  $(\gamma, T, k) \leftarrow\$ \mathsf{commit}(\mathsf{ST}[\mathsf{ID}], \vec{P})$
3 :  $t \leftarrow \mathsf{ep}[\mathsf{ID}]$
4 :  $\mathsf{T}[t, \text{‘}comm', \text{++}\mathsf{comm\text{–}ctr}] \leftarrow T$
5 :  $\mathsf{K}[t+1] \leftarrow k; \mathsf{ST}[\mathsf{ID}] \leftarrow \gamma$
```

```
$\mathcal{O}^{\mathsf{Deliver}}(\mathsf{ID}, t, c)$
1 :  if $\mathsf{C}[t] \in \{c, -1\}, \mathsf{C}[t] \leftarrow c$
2 :  else return
3 :  $T \leftarrow \mathsf{T}[t, \text{‘}comm', c]$
4 :  $\gamma \leftarrow \mathsf{proc}(\mathsf{ST}[\mathsf{ID}], T)$
5 :  if $\mathsf{ID} \notin \gamma.\mathsf{usrs} \wedge \mathsf{ID} \notin \gamma.\mathsf{cbts}$
6 :    $\mathsf{ep}[\mathsf{ID}] \leftarrow -1$
7 :  else
8 :    $\mathsf{ep}[\mathsf{ID}] \leftarrow t + 1$
9 :    if $\gamma.k \neq \perp$
10 :      $\mathsf{K}[t+1] \leftarrow \gamma.k$
11 :    $\mathsf{usrs}[t+1] \leftarrow \gamma.\mathsf{usrs}$
12 :    $\mathsf{cbts}[t+1] \leftarrow \gamma.\mathsf{cbts}$
13 :  $\mathsf{ST}[\mathsf{ID}] \leftarrow \gamma$
```

Figure B.1: Shared oracles for security games for our-CGKA. Highlighted code denotes the modification apart from original definition.

Our game-based security definition is based on Alwen et al. [3] and and a refined version defined in [8], which captures group key indistinguishability against adversaries with access to group membership and control messages. Figure B.1 illustrates the fundamental oracles for the adversaries to manipulate a group, such as group creation, making

doi:10.6342/NTU202401660

proposals and commits, delivering a control message, etc. These oracles are accessible to external adversaries or chatbot adversaries, reflecting the potential for attackers to compromise the service provider or individual chatbots.

We extend the definition to align with our extended syntax of CGKA, as highlighted in Figure B.1. Specifically, the game additionally records the chatbot member given for each epoch, and the proposal now takes an additional list of chatbot ids. An important modification is at line 9 of Deliver oracle, where the group key is updated only when non-empty key is obtained. This is because the selective message access feature prevents unauthorized chatbot from acquiring new keys. In that case, the variable $\gamma.k$ is set to $\perp$, denoting an absence of key access.

B.1 Security Game for External Adversaries

We follow Alwen et al. [3] to model group key indistinguishability against external adversaries with key compromise capabilities.

B.1.1 Key Indistinguishability Game

The game KIND, as illustrated in Figure B.2, is parameterized with an adversary $\mathcal{A}$, a protocol $\mathsf{Our–CGKA}$, and a cleanness predicate $\mathsf{C}_{\mathsf{CGKA}}$. Our game is based on the one defined in Alwen et al. [3]. The challenger first selects a random bit b. The adversary $\mathcal{A}$ may then query the shared oracles depicted in Figure B.1 and the oracles in Figure B.2. Finally, the adversary outputs the guess b', and wins the game if $b = b'$.

Using the queries, the adversary can manipulate the group and make state compro-

doi:10.6342/NTU202401660

mises. For each epoch t, the adversary can choose to either reveal the group key k or make a challenge, i.e. distinguish between the true group key with a random value.

```
$\mathsf{KIND}^{\mathcal{A}}_{\mathsf{Our\text{-}CGKA},\mathsf{C}_{\mathsf{CGKA}}}(1^\lambda)$
1 :  $b \leftarrow\$ \{0,1\}$
2 :  $\mathsf{exp}[\cdot] \leftarrow 0; \mathsf{chall}[\cdot] \leftarrow \mathsf{false}$
3 :  $b' \leftarrow\$ \mathcal{A}^{O}(1^\lambda)$
4 :  **assert** $\mathsf{C}_{\mathsf{CGKA}}$
5 :  **return** $1_{b=b'}$

$\mathcal{O}^{\mathsf{Expose}}(\mathsf{ID})$
1 :  $\mathsf{exp}[\mathsf{ID}, ++\mathsf{exp\text{-}ctr}] \leftarrow \mathsf{ep}[\mathsf{ID}]$
2 :  **return** $\mathsf{ST}[\mathsf{ID}]$

$\mathcal{O}^{\mathsf{Reveal}}(t)$
1 :  **assert** $\mathsf{K}[t] \neq \perp) \wedge \neg\mathsf{chall}[t]$
2 :  $\mathsf{chall}[t] \leftarrow \mathsf{true}$
3 :  **return** $\mathsf{K}[t]$

$\mathcal{O}^{\mathsf{Challenge}}(t)$
1 :  **assert** $(\mathsf{K}[t] \neq \perp) \wedge \neg\mathsf{chall}[t]$
2 :  $\mathsf{chall}[t] \leftarrow \mathsf{true}$
3 :  **if** $b = 0$ **return** $\mathsf{K}[t]$
4 :  **if** $b = 1$ **return** $r \leftarrow \{0,1\}^\lambda$
```

Figure B.2: Chatbot key indistinguishability KIND game for our-CGKA, parameterized by the cleanness predicate $\mathsf{C}_{\mathsf{CGKA}}$.

The cleanness predicate $\mathsf{C}_{\mathsf{CGKA}}$ is designed to exclude trivial attacks, such as immediately challenging a key after a compromise without subsequent updates. We adopt the predicate defined in [8], which checks for every user member ID in the group, and for each pair of compromise and challenge, it returns true in the two cases: (1) there is an update, add, or removal between the compromise and challenge, capturing PCS, or (2) the challenge occurs before the compromise, capturing FS. Any other scenario is considered invalid by the predicate.

The helper function $\mathsf{tExp}(\mathsf{ID}, \mathsf{ctr})$ returns $\mathsf{exp}[\mathsf{ID}, \mathsf{ctr}]$ if $\exists k : q_k = \mathcal{O}^{\mathsf{Expose}}(\mathsf{ID})$ or -1 otherwise. The helper function $hasupd(\mathsf{ID}, T)$ returns true if the commit corresponding to the control message T contains proposals that affects ID by add, removal, or updates.

A $\mathsf{Our\text{-}CGKA}$ scheme is defined to be secure if the adversary's advantage to win the game is negligible. Let (t, q) adversary denote an adversary with running time t and having at most q query accesses, we define the security of CGKA as follows.

doi:10.6342/NTU202401660

$$\begin{aligned}\boxed{\mathsf{C}_{\mathsf{CGKA}}} : \ & \forall(i, \mathsf{ID}, \mathsf{ctr} \in (0, \mathsf{exp{-}ctr}]) : q_i = \mathcal{O}^{\mathsf{Challenge}}(t_i^*),\\ & (\mathsf{ID} \notin \mathsf{usrs}[t_i^*])\vee\\ & (\exists(t_i, c) : \mathsf{tExp}(\mathsf{ID}, \mathsf{ctr}) < t_i \leq t_i^* \wedge\\ & \mathsf{hasUpd}(\mathsf{ID}, \mathsf{T}[t_i, `com', c]) \wedge (\mathsf{C}[t_i] = c)\vee\\ & (t_i^* < \mathsf{tExp}(\mathsf{ID}, \mathsf{ctr}))\end{aligned}$$

Figure B.3: Cleanness predicate for key indistinguishability, where the adversary makes queries $q_1, \ldots, q_n$.

Definition 5 (CGKA security of Our–CGKA)**.** *A our-CGKA protocol is* (t, q, ϵ)-KIND-*secure w.r.t the cleanness predicate* $\mathsf{C}_{\mathsf{CGKA}}$ *if for any* (t, q) *adversary* $\mathcal{A}$*,*

$$\left|\Pr\left[\mathsf{KIND}^{\mathcal{A}}_{\mathsf{Our{-}CGKA},\mathsf{C}_{\mathsf{CGKA}}}(1^\lambda) = 1\right] - \frac{1}{2}\right| \leq \varepsilon.$$

B.2 Security Game for Chatbot Adversaries

In order to model the capabilities of a chatbot adversary, we introduce a new oracle, Setup, which discloses the states and keys of a chatbot identified by CID. The oracle captures the fact that the adversary acts as a chatbot CID and is accessible to the following two security games for chatbot adversaries. Note that the oracle Setup can be invoked multiple times, which captures that there can be multiple malicious chatbots.

B.2.1 Sender Anonymity Game

The game Anon, as illustrated in Figure B.4, is parameterized with an adversary $\mathcal{A}$, a protocol Our–CGKA. We use sender indistinguishability to model sender anonymity for group messaging, similar to the definition used in [17, 27]. Similar to the KIND game, a random bit b is chosen initially, and the adversary can query the oracles multiple times. Finally, the adversary outputs the guess b', and wins the game if $b = b'$.

doi:10.6342/NTU202401660

The challenge oracle allows the adversary to select two user members, denoted $\mathsf{ID}_0, \mathsf{ID}_1$, along with a chatbot CID within the group. The oracle then triggers an key update by the chosen user member id_b, which is accessible to the chatbot CID. The adversary finally receives the group key k, the control message T, and the chatbot's state $\mathsf{ST}[\mathsf{CID}]$. This output simulates the information that a chatbot would receive following a group key update

```
Anon^A_{Our–CGKA}(1^λ)
1 :  $b \leftarrow\$ \{0,1\}$
2 :  $b' \leftarrow\$ \mathcal{A}^{O}(1^\lambda)$
3 :  **return** $1_{b=b'}$
```

```
$\mathcal{O}^{\mathsf{Challenge}}(\mathsf{ID}_0, \mathsf{ID}_1, \mathsf{CID})$
1 :  **assert** $\mathsf{ep}[\mathsf{ID}_0], \mathsf{ep}[\mathsf{ID}_1], \mathsf{ep}[\mathsf{CID}] \neq -1$
2 :  $(\gamma, P) \leftarrow\$ \mathsf{prop}(\mathsf{ST}[\mathsf{ID}_b], \perp, \mathsf{CID}, \mathsf{upd})$
3 :  $(\gamma, T, k) \leftarrow\$ \mathsf{commit}(\gamma, (P))$
4 :  $t \leftarrow \mathsf{ep}[\mathsf{ID}_b]$
5 :  $\mathsf{T}[t, \text{`}comm', \text{++comm–ctr}] \leftarrow T$
6 :  $\mathsf{K}[t+1] \leftarrow k; \mathsf{ST}[\mathsf{ID}_b] \leftarrow \gamma$
7 :  **return** $k, T, \mathsf{ST}[\mathsf{CID}]$
```

Figure B.4: Sender anonymity game (Anon) for our-CGKA.

Definition 6 (Sender Anonymity of Our–CGKA)**.** *A our-CGKA protocol is* (t, q, ϵ)-Anon-*secure if for any* (t, q) *adversary* $\mathcal{A}$*,*

$$\left| \Pr\left[\mathsf{Anon}^{\mathcal{A}}_{\mathsf{Our–CGKA}}(1^\lambda) = 1\right] - \frac{1}{2} \right| \leq \varepsilon.$$

B.2.2 Selective Message Access Game

The game SMA, as illustrated in Figure B.6, is parameterized with an adversary $\mathcal{A}$, a protocol Our–CGKA, and a predicate $\mathsf{C}_{\mathsf{CGKA}}$. The game aims to model key indistinguishability against chatbot adversaries unauthorized to the keys. So the game is almost identical to the KIND game, the only difference is that the Expose oracle now can only expose the state of a chatbot, and we don't record the epochs of exposures anymore, since the adversaries are allowed to be the chatbots at anytime.

doi:10.6342/NTU202401660

```
$\mathsf{SMA}^{\mathcal{A}}_{\mathsf{Our\text{–}CGKA}}(1^\lambda)$
1 :  $b \leftarrow\!\!\$ \{0,1\}$
2 :  $\mathsf{chall}[\cdot], \mathsf{exp}[\cdot] \leftarrow \mathsf{false}$
3 :  $b' \leftarrow\!\!\$ \mathcal{A}^{O}(1^\lambda)$
4 :  **assert** $\mathsf{C}_{\mathsf{CGKA\text{–}cbt}}$
5 :  **return** $1_{b=b'}$

$\mathcal{O}^{\mathsf{Reveal}}(t)$
1 :  **assert** $(\mathsf{K}[t] \neq \perp) \wedge \neg \mathsf{chall}[t]$
2 :  $\mathsf{chall}[t] \leftarrow \mathsf{true}$
3 :  **return** $\mathsf{K}[t]$

$\mathcal{O}^{\mathsf{Expose}}(\mathsf{CID})$
1 :  $\mathsf{exp}[\mathsf{CID}] \leftarrow \mathsf{true}$
2 :  **return** $\mathsf{ST}[\mathsf{CID}]$

$\mathcal{O}^{\mathsf{Challenge}}(t)$
1 :  **assert** $(\mathsf{K}[t] \neq \perp) \wedge \neg \mathsf{chall}[t]$
2 :  $\mathsf{chall}[t] \leftarrow \mathsf{true}$
3 :  **if** $b = 0$ **return** $\mathsf{K}[t]$
4 :  **if** $b = 1$ **return** $r \leftarrow \{0,1\}^\lambda$
```

Figure B.5: Selective message access (SMA) game for our-CGKA, parameterized by the cleanness predicate $\mathsf{C}_{\mathsf{CGKA\text{–}cbt}}$.

The adversary can compromise chatbots' states to act as a chatbot. Our objective is to secure keys derived from updates that do not involve these compromised chatbots. To prevent trivial attacks, such as those where a chatbot CID is compromised and then used to challenge an authorized key, we employ a cleanness predicate $\mathsf{C}_{\mathsf{CGKA\text{–}cbt}}$, as defined as follows, where $\mathsf{hasUpdCbt}(\mathsf{CID}, t, c)$ returns true if $\mathsf{T}[t, \text{‘}comm', c]$ contains any proposal P with type upd–cbt and the argument $\vec{\mathsf{CID}}$ includes CID.

$$\boxed{\mathsf{C}_{\mathsf{CGKA\text{–}cbt}}} : \forall(i, \mathsf{CID}) : q_i = \mathcal{O}^{\mathsf{Challenge}}(t_i^*), \\ (\neg\, \mathsf{exp}[\mathsf{CID}]) \vee \\ (\nexists c : \mathsf{hasUpdCbt}(\mathsf{CID}, t_i^* - 1, c)) \wedge \\ (\mathsf{C}[t_i] = c)$$

Figure B.6: Cleanness predicate for selective message access, where the adversary makes queries $q_1, \ldots, q_n$.

Definition 7 (Selective Message Access of Our–CGKA). *A our-CGKA protocol is* (t, q, ϵ)-SMA-*secure w.r.t the cleanness predicate* $\mathsf{C}_{\mathsf{CGKA\text{–}cbt}}$ *if for any* (t, q) *adversary* $\mathcal{A}$,

$$\left| \Pr\left[\mathsf{SMA}^{\mathcal{A}}_{\mathsf{Our\text{–}CGKA}, \mathsf{C}_{\mathsf{CGKA\text{–}cbt}}}(1^\lambda) = 1\right] - \frac{1}{2} \right| \leq \varepsilon.$$

doi:10.6342/NTU202401660