Guardianship in Group Key Exchange for Limited Environments

Elsie Mestl Fondevik[*1,2], Britta Hale[†3] and Xisen Tian[‡3]

[1]Kongsberg Defence & Aerospace, Norway[*]
[2]Norwegian University of Science and Technology, Trondheim, Norway
[3]Naval Postgraduate School, USA [†‡]

Abstract

Post-compromise security (PCS) has been a core goal of end-to-end encrypted messaging applications for many years, both in one-to-one continuous key agreement (CKA) and for groups (CGKA). At its essence, PCS relies on a compromised party to perform a key update in order to 'self-heal'. However, due to bandwidth constraints, receive-only mode, and various other environmental demands of the growing number of use cases for such CGKA protocols, a group member may not be able to issue such updates. In this work, we address the issue of devices functioning in limited mode through the introduction of *guardianship*, where a designated guardian can perform key updates on the behalf of its paired edge device. We introduce a Guardianship PCS (GPCS) security, and provide an associated security experiment. We investigate various architectural designs in the pursuit of GPCS, provide constructions and security analyses, and describe trade-offs.

1 Introduction

Instant messaging has ignited research on achieving forward secrecy (FS) and post-compromise security (PCS) guarantees. Formally, this is done via a continuous key agreement (CKA) between parties wherein keys are ratcheted from one epoch to the next. Thus, past keys remain secure even under state reveal at a later epoch (i.e. FS). If new keying material is introduced at each ratchet, CKA can also provide PCS for "self-healing" of the connection after an honest update from the compromised device. CKA has extended to continuous group key agreement (CGKA) for multiple devices, with the Messaging Layer Security (MLS) [9] being the first such standardized group variant.

In group communications, asynchronous support is especially relevant since some users may not be able to send messages due to environmental constraints. In such cases, long periods of user inactivity can undermine the group's security, leading to the current recommendation being to evict them such silent users [9]. This policy discourages devices or users routinely operating in limited environments (e.g. remote sensors or users in disaster zones) to participate in a CGKA protocol.

While extensive work exists on MLS and CGKAs, research on methods for addressing PCS security when devices have limited ability to send updates is lacking. One intuitive solution for this is if the offline user "edge" device is paired with another "guardian" device of the user's that could update on its behalf. For instance, if user Alice has a mobile device, laptop, tablet, smart watch, etc., one of those could be left in a secure and reliable bandwidth location allowing it take on the key update action. We call the concept of PCS key updates on behalf of another device *Guardian PCS (GPCS)*. Of course, GPCS raises questions about what information a guardian has or does not have access to, whether the guardian can impersonate the edge device to the group (or vice versa), whether the other group members have awareness of the guardian existence and its relationship to the edge, and many more core security considerations. In this work, we delve into the guardianship space and initialize a study of the trade-offs, construction options, and how architectural design considerations affect the end security.

2 Motivation and Related Work

This work is based in continuous key agreement protocols, which has historically built out from one-to-one key agreement to also include group key agreement protocols. End-to-end encryption [19] is a foundational goal and assumption in all cases we consider.

2.1 Continuous Key Agreement and Security

The Signal protocol [26] marked a pivot point in the design of secure messaging due to a variety of security properties

[*]elsie.fondevik@kongsberg.com
[†]britta.hale@nps.edu
[‡]xisen.tian1@nps.edu

supported beyond confidentiality and data authenticity. Research has since expanded, with a wide breadth of work now available on the design and analysis on such CKA protocols. Unlike session-based protocols such as TLS [31], QUIC [20], IPSec [18], CKA protocols offer a single, long-lived session with unspecified termination. Instead of periodic handshakes to establish keying material, CKA continuously evolve the secret state through asynchronous *updates* of new keying material. Due to the asynchronicity of the updates, CKA has the ability to support offline devices and even devices in receive-only mode. This has led to CKA use in messaging applications such as Signal [28] and Whatsapp [11], and Facebook Messenger [27], and a variety of modeling techniques for assessing security [2, 14, 16, 17, 21, 22, 30, 32], as well as vulnerabilities [1].

The pursuit of FS and PCS in group messaging protocols can sometimes be at odds with efficiency. After all, a single long-lived shared group state with no updates to the keying material supports good application performance under limited bandwidth or computational power – it also comes at the cost of FS and PCS security.[1] This has led to an effort for the design and standardization of a CGKA protocol, Messaging Layer Security (MLS) [10], and has since extensive analysis throughout its development cycle [4, 5, 8, 12, 13], including under continuous group key agreement (CGKA) model [7].

2.2 Guardianship vs Ghosts

One may wonder about the distinction between guardianship and ghost users as have been proposed [25] and are often controversial [29]. First, a fundamental goal of guardianship is preservation of the FS and PCS guarantees often found in current end-to-end encrypted messaging protocols (this work notes intuitive guardianship designs that would break this foundation and should be avoided). Unlike under ghost users, where a third party is silently added to the group, guardianship is an additional device under the current user's ownership, with varying degrees of visibility to other group members. The guardian's role is to enforce strong FS and PCS guarantees even if its corresponding edge is unable to send updates, and as such there is a fundamental assumption on the guardian of being held in a more secure location with less access and risk of compromise. *More* access to the guardian – such as through various administrators on a ghost user, would fundamentally undermine potential guardianship benefits. Thus, an effective guardian is an asset under the end-user's control.

[1] It should be noted that traditional definitions for PCS do not account for authentication issues or impersonation from the compromise of signature keys [15], instead focusing on *confidentiality* properties only. We similarly focus on confidentiality in this paper, but provide an overview comparision of entity authenticity properties in Section 8.

2.3 Contributions

We initialize a study of guardianship and its use within continuous group key agreement. Specifically, we

- introduce a formal definition for Guardian Continuous Group Key Agreement (GCGKA) and associated security model GCGKA.
- provide a cross comparison of the design space of guardian extension protocols for the MLS.
- analyze selected GCGKA protocols over MLS for architectural design considerations and GCGKA security.

3 Prerequisites

We build upon the basis of a continuous group key agreement (CGKA) protocol, and leverage the underlying notation from [7] as found in Definition 1. The eventual constructions will use MLS as a CGKA example with guardianship capabilities added to it.

Definition 1 (Continuous Group Key Agreement, adapted from [7] for clarity)**.** *A continuous group key-agreement (CGKA) scheme* $CGKA = (\mathsf{init}, \mathsf{create}, \mathsf{add}, \mathsf{rem}, \mathsf{upd}, \mathsf{proc})$ *consists of the following algorithms:*

- *Initialize:* init *takes as input a group member identity,* ID*, and outputs an initial shared group state* $\gamma[\mathsf{ID}]$*.*
- *Create Group:* create *takes as input a state* $\gamma[\mathsf{ID}]$ *and a list of identities for group members* $G = (\mathsf{ID}_1, \ldots, \mathsf{ID}_n)$*, and outputs a new state* $\gamma[\mathsf{ID}]$ *and a welcome message Welcome.*
- *Add Group Member:* add *takes a shared group state* $\gamma[\mathsf{ID}]$ *and ID,* ID*, and outputs a new state* $\gamma[\mathsf{ID}]$*, welcome message Welcome, and control message (public update value)* T*.*
- *Remove Group Member:* rem *takes a state* $\gamma[\mathsf{ID}]$ *and an ID,* ID*, and outputs a new group state* $\gamma[\mathsf{ID}]$ *and a control message (public update value)* T*.*
- *Update Generation:* upd *takes a shared group state,* $\gamma[\mathsf{ID}]$*, and outputs a new state* $\gamma[\mathsf{ID}]$ *and a control message (public update value)* T*.*
- *Process Received Update:* proc *takes a state* $\gamma[\mathsf{ID}]$ *and a control message (public update value)* T *and outputs a new state* $\gamma[\mathsf{ID}]$ *and an update secret* I*.*

Definition 2 (CGKA Security (adapted from [7]))**.** *Let* Π *be a CGKA protocol and let* $\mathcal{A}$ *be a PPT adversarial algorithm against* Π *as defined in* gray *in Figures 2, 3 and 4b. We define the adversarial advantage of* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{CGKA}_{\Pi-\mathcal{A}}(\lambda) = \left| \Pr\left[Exp^{CGKA}_{\Pi-\mathcal{A}}(\lambda) \right] - \frac{1}{2} \right|$$

and say that the protocol Π *is* CGKA-secure *if* $\mathsf{Adv}^{CGKA}_{\Pi-\mathcal{A}}(\lambda)$ *is negligible for all* $\mathcal{A}$*.* [2]

[2] For CGKA [7] the safety predicate Figure 4b aligns to when $X \in \{FS, PCS\}$ and $\mathfrak{l} = \mathsf{ID}$. [7]

Some of our constructions also utilize the continuous key agreement (CKA) protocol for a secure pairwise channel between the guardian and edge. Figure 1 is provided as a reference for those schemes using CKA.

Definition 3 (Continuous Key Agreement (CKA) [3])**.** *A continuous-key-agreement (CKA) scheme is a quadruple of algorithms CKA = CKA-Init-A, CKA-Init-B, CKA-S, CKA-R), where*

- *CKA-Init-A (and similarly CKA-Init-B) takes a key k and produces an initial state $\gamma^A \leftarrow$ CKA-Init-A(k) (and γ^B).*
- *CKA-S takes a state γ, and produces a new state, message, and key $(\gamma', T, I) \leftarrow\$$ CKA-S(γ), and*
- *CKA-R takes a state γ and message T and produces a new state and a key $(\gamma', I) \leftarrow$ CKA-R(γ, T).*

Denote by $\mathcal{K}$ the space of initialization keys k and by I the space of CKA keys I.

Definition 4 (CKA Security (adapted from [6])**.** *Let CKA be a continuous key agreement protocol, and let $t^* \in \mathbb{N}$ be an epoch index and $\Delta_{CKA} \in \mathbb{N}$ be the number of epochs until an epoch no longer contains secret information pertaining to a challenge. For a PPT algorithm $\mathcal{A}$, we define the advantage of $\mathcal{A}$ in the CKA security experiment (see Figure 1) to be*

$$\mathsf{Adv}^{CKA,t^*,\Delta_{CKA}}_{CKA-\mathcal{A}}(\lambda) = \left| \Pr[Exp^{CKA,t^*,\Delta_{CKA}}_{CKA-\mathcal{A}}(\lambda) = 1] - \frac{1}{2} \right| .$$

We say that CKA is CKA-secure if, for all $\mathcal{A}$, $\mathsf{Adv}^{CKA,t^*,\Delta_{CKA}}_{CKA-\mathcal{A}}(\lambda)$ is negligible in the security parameter λ.

Our constructions also utilize a Key Derivation Function. Here we provide the definition for PRF security of the KDF.

Definition 5 (Key Derivation Function (KDF) (adapted from [24]))**.** *A* key derivation function*, $KDF(sk, \mathtt{id}) \rightarrow sk'$, is a pseudorandom function $\mathcal{K} \times \mathcal{N} \rightarrow \mathcal{K}$ that takes as input original keying material sk and an optional key identifier* id *and outputs a key sk'.*

4 Terminology and GCGKA Definition

The general intuition behind GCGKA is to to expand existing group key exchange protocols to allow for users to enter an operational mode that potentially limits their ability to send protocol updates – for example due to bandwidth, power, or other limitations – while still achieving a degree of security that they could not otherwise have. We treat the underlying group key exchange as a CGKA blackbox, without modification, and rather enhance it to allow guardians to update on behalf of another specified user. As a requirement, guardianship should not reduce the security guarantees from the blackbox group protocol core.

4.1 Terminology

Edge Device: an *edge* device is an original user equipment device. Such a device may operate in receive-only mode or in another limited fashion such that sending regular keying updates is impractical or even impossible.

Guardian: the *guardian* device operates from a secured space with reliable network access. The intent is for the guardian device to be paired with, and provide keying updates on behalf of the edge device. This supports forward secrecy in the event the edge device is compromised. When the edge device is in an active state that allows for it to perform keying updates of its own, the guardian device may be placed in offline mode.

The exact forward secrecy properties, as well as the guardian's access to messages, will be explored later. Specifically, these properties depend on the guardian/edge architecture choice as well as distribution service.

Edge Device Operational Modes An edge device has two modes of operation. Before an edge device enters a new state the MLS delivery service (DS), which facilitates group state consensus by ensuring in-order delivery of MLS messages, must be informed. The operational states are:

Online mode: The edge device is available and running the group protocol as per specification. In this state it does not have need of a guardian.

Limited mode: The edge device has the ability to *receive* application messages and key update messages, but it may not preform its own key updates. Depending on environmental situation, an edge device may still be allowed to send application messages while in this mode. In this state, a guardian may send updates on behalf of the edge device.

Guardian Operational Modes A guardian may be in one of two modes as well, contingent on the mode of the edge.

Offline mode: When an edge device is online the guardian may be set to be inactive (depending on the guardianship construction).

Online mode: When the edge device enters limited mode, it becomes reliant on a guardian. Therefore the guardian status must be set to online in order to send key updates.

4.2 Notation and State Variables

Independent of construction a GP consists of an edge device and a guardian that interchangeably access the underlying group to update the group session key. These access point(s) into the underlying group used by the guardian and edge device can be viewed as a group member in the underlying group and has its own unique ID inherited from CGKA. We call this CGKA group member node an *anchor*[3]. In the experiment and definition we notationally differentiate between anchor ID's (ID) and guardian/edge device ID's (id_G/id_E). Depending on protocol design, the guardian and edge device may share an anchor or they may have distinct anchors.

[3]For example, in MLS, an anchor would be a leaf node in the MLS tree.

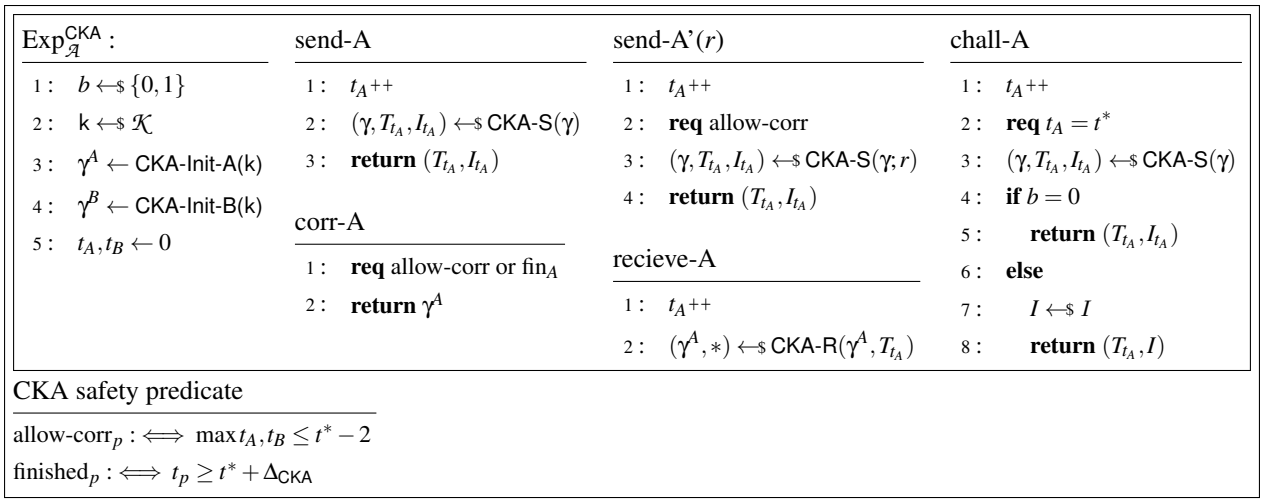


Figure 1: The CKA experiment from [3] shows how to secure a pair-wise channel against probabilistic polynomial time (PPT) adversary, $\mathcal{A}$, given a safety predicate that prevents trivial wins.

To the underlying protocol an anchor is indistinguishable from a regular group member. The anchor maintains the same variables as any CGKA group member: an identity ID and a state $\gamma[\mathsf{ID}]$. However, the anchor is a logical construct node vs. a real device; its state is accessed and maintained by the edge device and/or guardian. Meanwhile, the edge device and guardian each consist of an identity, id_E and id_G respectively, together with an expanded state $S[id_E]$ and $S[id_G]$ respectively. For an edge device id_E with anchor ID_E, the state $S[id_E] = (\gamma[id_E], \gamma[\mathsf{ID}_E])$ is a two tuple consisting of the state of edge device-specific information together with the state of the anchor node. This inherently includes the associated signing key of the anchor. If the edge device does not have direct access to the anchor, the $\gamma[\mathsf{ID}_E]$ component is set to ε. The guardian state is similarly constructed using the guardian's id, e.g., $S[id_G] = (\gamma[id_G], \gamma[\mathsf{ID}_G])$.

Moreover, the edge and guardian sessions contain encoded information. For an edge or guardian id, $mode[id]$ is a binary flag specifying the instances' operational mode: $\{isOnline, isOffline\}$ for guardians and $\{isOnline, isLimited\}$ for edges. The variable $anchor[id]$ stores the id's anchor ID. Additionally have the session state variable $getGuardian[id]$ and $getEdge[id]$, which specifies the guardian id or edge of a particular id respectively.

4.3 GCGKA Protocol Definition

A guardian protocol consist of eight algorithms in addition to those inherited form CGKA. The first three algorithms focus on the addition and removal of guardian or edge devices. In the case of addition, it is assumed that anchor(s), if needed, already exist. If a user wishes to join the group directly as a edge device with a guardian then the edge device will first need to become a member of the underlying group using CGKA function add before initiating the guardianship protocol to include the guardian. When exiting the guardianship protocol (i.e., returning to normal, non-guardianship membership), the guardian will be removed. The edge device will, in this case, be transformed into a regular group member. If the edge device is removed, both guardian and all corresponding anchors will be removed from the underlying group.

The next two algorithms, **enterLimitedMode** and **exitLimitedMode**, toggle between the operational modes. These functions affect who is authorised to preform updates in the underlying group.

The final three algorithms are used to preform and process key updates. Updates may either be issued to the underlying group or between guardian and edge device. Similarly, processing is split in two; a received message may either be in the format of the underlying group or a GP proprietary format. Depending on the type, the input is processed accordingly.

Formally we define a guardian protocol as follows.

Definition 6. *Let Π be a CGKA protocol according to Definition 1. We extend Π to a guardian CGKA protocol, GCGKA, by introducing the following algorithms:*

enterGship In: $\gamma[\mathsf{ID}_E], \gamma[\mathsf{ID}_G]$ ***Out:*** $id_E, S[id_E], id_G, S[id_G]$
This algorithm takes the anchor states for an edge $S[id_E]$ and guardian $S[id_G]$ as input and returns ids for the added edge and guardian devices together with their initial states. It is required that no edge or guardian is currently appended to the anchor ids. If the operation could not be preformed, an error symbol $\perp$ is returned.

exitGship In: id_E, $S[id_E]$, id_G ***Out:*** $\gamma[\mathsf{ID}_E]$, T
This algorithm takes as input an edge id id_E, the state of the edge device $S[id_E]$, and a guardian id id_G, and returns an updated anchor CGKA state $\gamma[\mathsf{ID}_E]$ and a control message T. If $getGuardian[id_E] \neq id_G$ or $getEdge[id_G] \neq id_E$, the algorithm outputs an error symbol $\perp$. The algorithm terminates the running of guardianship protocol. If the

edge and guardian have separate anchors, the guardian anchor is removed from the group via a call to the CGKA rem *algorithm.*

removePair ***In:*** $\gamma[ID]$, ID_E, ID_G, ***Out:*** $\gamma[ID]$, T_1, T_2

The algorithm takes as input the underlying CGKA group leader's state, $\gamma[ID]$, and the edge device's and guardian's anchor id's, ID_E and ID_G respectively. If the edge device and guardian share an anchor, $ID_E = ID_G$. If $getGuardian[id_E] \neq id_G$ or $getEdge[id_G] \neq id_E$, the algorithm outputs an error symbol $\perp$. The algorithm calls the CGKA rem *remove algorithm twice:* $\mathsf{rem}(\gamma[ID], ID_G) \rightarrow (\gamma[ID]', T_1)$ *and then* $\mathsf{rem}(\gamma[ID]', ID_E) \rightarrow (\gamma[ID]'', T_2)$*, removing both anchors from the underlying CGKA group. An updated CGKA state ($\gamma[ID] \leftarrow \gamma[ID]''$) together with two CGKA control messages is returned as output. If the edge device and guardian share the same anchor, then* rem *is only called once and T_2 is set to* ε.

enterLimitedMode ***In:*** id_E, id_G ***Out:*** ε

This algorithm takes as input the id of an edge device and its guardian, id_G. If $getGuardian[id_E] \neq id_G$ or $getEdge[id_G] \neq id_E$, the algorithm outputs an error symbol $\perp$. The algorithm sets $mode[id_E] \leftarrow isOffline$ to put the edge device into limited mode while $mode[id_G] \leftarrow isOnline$ sets guardian as active.

exitLimitedMode ***In:*** id_E, id_G ***Out:*** ε

This algorithm takes as input the id of an edge device, id_E, and its guardian, id_G. If $getGuardian[id_E] \neq id_G$ or $getEdge[id_G] \neq id_E$, the algorithm outputs an error symbol $\perp$. The algorithm sets $mode[id_E] \leftarrow isOnline$, putting it online, while $mode[id_G] \leftarrow isOffline$ the guardian is set to offline mode (when possible [4]).

updateE ***In:*** id_E, $S[id_E]$ ***Out:*** $S[id_E]$, T

The algorithm takes as input an edge device id, id_E, and its corresponding state $S[id_E]$. It checks that $mode[id_E] = isOnline$, outputting an error symbol $\perp$ if not, and otherwise outputs an updated edge device state together $S[id_E]$ with a control message T.

updateG ***In:*** id_G, $S[id_G]$ ***Out:*** $S[id_G]$, T

The algorithm takes as input a guardian id, id_G, and its corresponding state $S[id_G]$. It checks that $mode[id_G] = isOnline$, outputting an error symbol $\perp$ if not, and otherwise outputs an updated guardian state $S[id_G]$ together with a control message T.

processUpdGE ***In:*** $S[id]$, T ***Out:*** $S[id]$, I

The algorithm takes as inputs the state of an edge or guardian, $S[id]$, and a control message T. It returns an updated device state and a group update secret I. [5]

[4] If the guardian is identical to the anchor for id_E, i.e. $ID_E = id_G$ as in some possible constructions, it may not be possible to put it into offline mode.

[5] The algorithm can process control messages to update the internal state whether those are CGKA-based or through a dedicate guardian-edge channel.

5 GCGKA Security

The CGKA security model is used as a basis for our guardianship security model. We assume the underlying group key exchange protocol, on which guardian protocol builds upon, is defined according to Definition 1. Consequently, the guardian schemes we explore can be applied to a variety of concrete CGKA schemes. The *guardian protocol (GCGKA)* model is constructed in such a way that group members may individually choose to run the GCGKA protocol or the underlying CGKA based protocol. It is therefore especially important that any communication between GCGKA and the underlying protocol is done according to CGKA specification.

For GCGKA we retain the following four requirements from CGKA [7], and include an additional fifth requirement, *guardian post-compromise security (GPCS)*.

Correctness: All group members output the same update secret I in update epochs.

Privacy: The update secrets look random given the transcript of control messages.

Forward secrecy (FS): If the state of any group member is leaked at some point, all previous update secrets remain hidden from the attacker.

Post-compromise security (PCS): After every group member whose state was leaked performs an update, the update secrets become secret again.

Guardian post-compromise security (GPCS): After the guardian for any edge device whose state was leaked performs an update (that is processed by the group), update secrets become secret again.

Remark 1. *The guardianship architecture allows for a distinct forward secrecy feature in edge device limited mode. Namely, even if all edge devices are in limited mode and cannot issue updates – a mode normally expanding the FS vulnerability window – the guardian can still control and limit that vulnerability window by issuing updates. This may be done on some automated periodicity.*

5.1 Security Experiment

The security experiment for GCGKA can be found in Figure 2 and Figure 3. The security experiment encompasses that of CGKA, where parts taken directly from CGKA can be found in gray. The new additions are written in black.

Initialization. The protocol is initiated by setting up the variables used in the experiment. With the exception of a few new variables, the process is identical to the CGKA INIT query. The identity states $\gamma[ID]$ are instantiated with a call to CGKA init for a set of IDs. Like CGKA, every epoch ($epoch[]$) has an update leader (lead[]), update secret (I[]), and group members (**Group**[]). Because users can issue multiple updates and other actions (e.g. add/remove) within a single epoch, a counter (ctr[]) is used to track the actions for the update reconciliation process via the control messages (T) which

$\text{Exp}^{\mathbf{X}-\mathsf{GCGKA}}_{\Pi-\mathcal{A}}(\lambda)$:

$b \leftarrow\$ \{0,1\}$
$\forall \mathsf{ID} : \gamma[\mathsf{ID}] \leftarrow \mathsf{init}(\mathsf{ID})$
$\forall \mathsf{ID} : \mathbf{E}[\mathsf{ID}], \mathbf{G}[\mathsf{ID}] \leftarrow \varepsilon$
$lead[\cdot], \mathbf{I}[\cdot], \mathbf{Group}[\cdot], S[\cdot] \leftarrow \varepsilon$
$epoch[\cdot], ctr[\cdot] \leftarrow 0$
$D[\cdot] \leftarrow \mathsf{true}$
$chall[\cdot] \cdot \leftarrow \mathsf{false}$
$\mathtt{Pub}\ \ \mathbf{M}[\cdot] \leftarrow \varepsilon$

GCGKA-EnterGship$(\mathsf{ID}_E, \mathsf{ID}_G)$

$t \leftarrow epoch[\mathsf{ID}_E]$
req $(t > 0 \wedge (\mathsf{ID}_E, \mathsf{ID}_G) \in \mathbf{Group}[t])$
req $(\mathbf{E}[\mathsf{ID}_E] = \mathbf{E}[\mathsf{ID}_G] = \varepsilon)$
req $(\mathbf{G}[\mathsf{ID}_E] = \mathbf{G}[\mathsf{ID}_E] = \varepsilon)$
$id_E, \gamma[id_E], id_G, \gamma[id_G]$
  $\leftarrow$ **enterGship**$(\gamma[\mathsf{ID}_E], \gamma[\mathsf{ID}_G])$
$\mathbf{E}[\mathsf{ID}_E] \leftarrow id_E$
$\mathbf{G}[\mathsf{ID}_G] \leftarrow id_G$
$S[id_E] \leftarrow (\gamma[id_E], \gamma[\mathsf{ID}_E])$
$S[id_G] \leftarrow (\gamma[id_G], \gamma[\mathsf{ID}_G])$

GCGKA-ExitGship(id_E, id_G)

req $getGuardian[id_E] = id_G$
req $getEdge[id_G] = id_E$
$\mathsf{ID}_E \leftarrow anchor(id_E)$
$\mathsf{ID}_G \leftarrow anchor(id_G)$
$t \leftarrow epoch[\mathsf{ID}_E]$
req $(\mathbf{E}[\mathsf{ID}_E] = id_E) \wedge (\mathbf{G}[\mathsf{ID}_G] = id_G)$
$(\gamma[\mathsf{ID}_E], T) \leftarrow$ **exitGship**$(id_E, S[id_E], id_G)$
$\mathbf{G}[\mathsf{ID}_G], \mathbf{E}[\mathsf{ID}_E] \leftarrow \varepsilon$
$S[id_E], S[id_G] \leftarrow \varepsilon$
$c \leftarrow {+}{+}ctr[\mathsf{ID}_E]$
for $\mathsf{ID}'' \in \mathbf{Group}[t]$:
  $\mathbf{M}[t+1, \mathsf{ID}_E, \mathsf{ID}'', c] \leftarrow T$
if $T =$ **removed**$(t+1, \mathsf{ID}_G)$:
  $\mathbf{Group}[t+1, \mathsf{ID}_E, c] \leftarrow \mathbf{Group}[t] \setminus \{\mathsf{ID}_G\}$

GCGKA-ToggleLimitedMode(id_E):

$id_G \leftarrow getGuardian[id_E]$
req $(id_G \neq \varepsilon) \wedge (getEdge[id_G] = id_E)$
if $mode[id_E] = isOnline$:
  enterLimitedMode(id_E, id_G)
else
  exitLimitedMode(id_E, id_G)

createGroup$(\mathsf{ID}_0, \mathsf{ID}_1, \cdots, \mathsf{ID}_n)$

$t \leftarrow epoch[\mathsf{ID}]$
req $t = 0$
$c \leftarrow {+}{+}ctr[\mathsf{ID}_0]$
$(\gamma[\mathsf{ID}_0], Welcome)$
  $\leftarrow \mathsf{create}(\gamma[\mathsf{ID}_0], \mathsf{ID}_1, \cdots, \mathsf{ID}_n)$
for $i = 0, \cdots n$
  $\mathbf{M}[t+1, \mathsf{ID}_0, \mathsf{ID}_i, c] \leftarrow Welcome$
$\mathbf{Group}[t+1, \mathsf{ID}, c] \leftarrow \{\mathsf{ID}_0, \mathsf{ID}_1, \cdots, \mathsf{ID}_n\}$

AddUser$(\mathsf{ID}, \mathsf{ID}')$

$t \leftarrow epoch[\mathsf{ID}]$
req $t > 0 \wedge \mathsf{ID}' \notin \mathbf{Group}[t]$
req $\mathbf{E}[\mathsf{ID}'] = \mathbf{G}[\mathsf{ID}'] = \varepsilon$
$c \leftarrow {+}{+}ctr[\mathsf{ID}]$
$(\gamma[\mathsf{ID}], Welcome, T) \leftarrow \mathsf{add}(\gamma[\mathsf{ID}], \mathsf{ID}')$
$M[t+1, \mathsf{ID}, \mathsf{ID}', c] \leftarrow (Welcome, T)$
for $\mathsf{ID}'' \in \mathbf{Group}[t]$:
  $\mathbf{M}[t+1, \mathsf{ID}, \mathsf{ID}'', c] \leftarrow T$
$\mathbf{Group}[t+1, \mathsf{ID}, c] \leftarrow \mathbf{Group}[t] \cup \{\mathsf{ID}'\}$

GCGKA-Remove$(\mathsf{ID}, \mathsf{ID}')$

$T_1, T_2 \leftarrow \varepsilon$
$t \leftarrow epoch[\mathsf{ID}]$
req $t > 0 \wedge \mathsf{ID}' \notin \mathbf{Group}[t] > 0$
$c \leftarrow {+}{+}ctr[\mathsf{ID}]$
if $(\mathbf{E}[\mathsf{ID}'] \neq \varepsilon) \vee (\mathbf{G}[\mathsf{ID}'] \neq \varepsilon)$:
  if $\mathbf{E}[\mathsf{ID}'] \neq \varepsilon$
    $\mathsf{ID}_E \leftarrow \mathsf{ID}'$
    $\mathsf{ID}_G \leftarrow anchor[getGuardian[\mathbf{E}[\mathsf{ID}_E]]]$
  else if $\mathbf{G}[\mathsf{ID}'] \neq \varepsilon$
    $\mathsf{ID}_G \leftarrow \mathsf{ID}'$
    $\mathsf{ID}_E \leftarrow anchor[getEdge[\mathbf{G}[\mathsf{ID}_G]]]$
  $\gamma[\mathsf{ID}], T_1, T_2 \leftarrow$ **removePair**$(\gamma[\mathsf{ID}], \mathsf{ID}_E, \mathsf{ID}_G)$
  $\hat{\mathsf{ID}} \leftarrow \{\mathsf{ID}_E, \mathsf{ID}_G\}$
  $\mathbf{E}[\mathsf{ID}_E], \mathbf{G}[\mathsf{ID}_G] \leftarrow \varepsilon$
else :
  $(\gamma[\mathsf{ID}], T_1) \leftarrow \mathsf{rem}(\gamma[\mathsf{ID}], \mathsf{ID}')$
  $\hat{\mathsf{ID}} \leftarrow \{\mathsf{ID}'\}$
for $\mathsf{ID}'' \in \mathbf{Group}[t]$:
  $\mathbf{M}[t+1, \mathsf{ID}, \mathsf{ID}'', c] \leftarrow T_1$
  if $T_2 \neq \varepsilon$:
    $\mathbf{M}[t+1, \mathsf{ID}, \mathsf{ID}'', c+1] \leftarrow T_2$
$\mathbf{Group}[t+1, \mathsf{ID}, c] \leftarrow \mathbf{Group}[t] \setminus \{\hat{\mathsf{ID}}\}$
$ctr[\mathsf{ID}] \leftarrow ctr[\mathsf{ID}] + ||\hat{\mathsf{ID}}|| - 1$

GCGKA-EdgeUpdate(id_E)

$\mathsf{ID}_E \leftarrow anchor[id_E]$
req $\mathbf{E}[\mathsf{ID}_G] = id_E$
req $mode[id_E] = isOnline$
req $getGuardian[id_E] \neq \varepsilon$
$t \leftarrow epoch[\mathsf{ID}_E]$
req $t > 0$
$S[id_E], T \leftarrow$ **updateE**$(id_E, S[id_E])$
if $T \in T_{CGKA}$
  $c \leftarrow {+}{+}ctr[\mathsf{ID}_E]$
  for $\mathsf{ID}'' \in \mathbf{Group}[t]$
    $\mathbf{M}[t+1, \mathsf{ID}_E, \mathsf{ID}'', c] \leftarrow T$
  $\mathbf{Group}[t+1, \mathsf{ID}_E, c] \leftarrow \mathbf{Group}[t]$
else
  $c \leftarrow {+}{+}ctr[id_E]$
  $\mathbf{M}[t, id_E, id_E, c] \leftarrow T$
  $\mathbf{M}[t, id_E, id_G, c] \leftarrow T$

GCGKA-GuardianUpdate(id_G)

$\mathsf{ID}_G \leftarrow anchor[id_G]$
req $\mathbf{G}[\mathsf{ID}_G] = id_G$
req $mode[id_G] = isOnline$
$t \leftarrow epoch[\mathsf{ID}_G]$
req $t > 0$
$S[id_G], T \leftarrow$ **updateG**$(id_G, S[id_G])$
if $T \in T_{CGKA}$
  $c \leftarrow {+}{+}ctr[\mathsf{ID}_G]$
  for $\mathsf{ID}'' \in \mathbf{Group}[t]$
    $\mathbf{M}[t+1, \mathsf{ID}_G, \mathsf{ID}'', c] \leftarrow T$
  $\mathbf{Group}[t+1, \mathsf{ID}_G, c] \leftarrow \mathbf{Group}[t]$
else
  $c \leftarrow {+}{+}ctr[id_G]$
  $\mathbf{M}[t, id_G, id_E, c] \leftarrow T$
  $\mathbf{M}[t, id_G, id_G, c] \leftarrow T$

SendUpdate(ID)

req $\mathbf{E}[\mathsf{ID}] = \mathbf{G}[\mathsf{ID}] = \varepsilon$
$t \leftarrow epoch[\mathsf{ID}]$
req $t > 0$
$(\gamma[\mathsf{ID}], T) \leftarrow \mathsf{upd}(\gamma[\mathsf{ID}])$
for $\mathsf{ID}'' \in \mathbf{Group}[t]$
  $\mathbf{M}[t+1, ID, ID'', c] \leftarrow T$
$\mathbf{Group}[t+1, \mathsf{ID}, c] \leftarrow \mathbf{Group}[\mathrm{t}]$

no-del$(\mathfrak{l} = (\mathsf{ID} \vee id))$

$D[\mathfrak{l}] \leftarrow \mathsf{false}$

Figure 2: Oracles for the GCGKA security game for a scheme GCGKA = (enterGuardianship, exitGuardianship, removePair, enterSilentMode, exitSilentMode, updateED, updateGuard, processGP). Gray text indicates CGKA experiment syntax and black indicated new or altered components. Algorithms and sets are indicated in bold font and variables in italics.

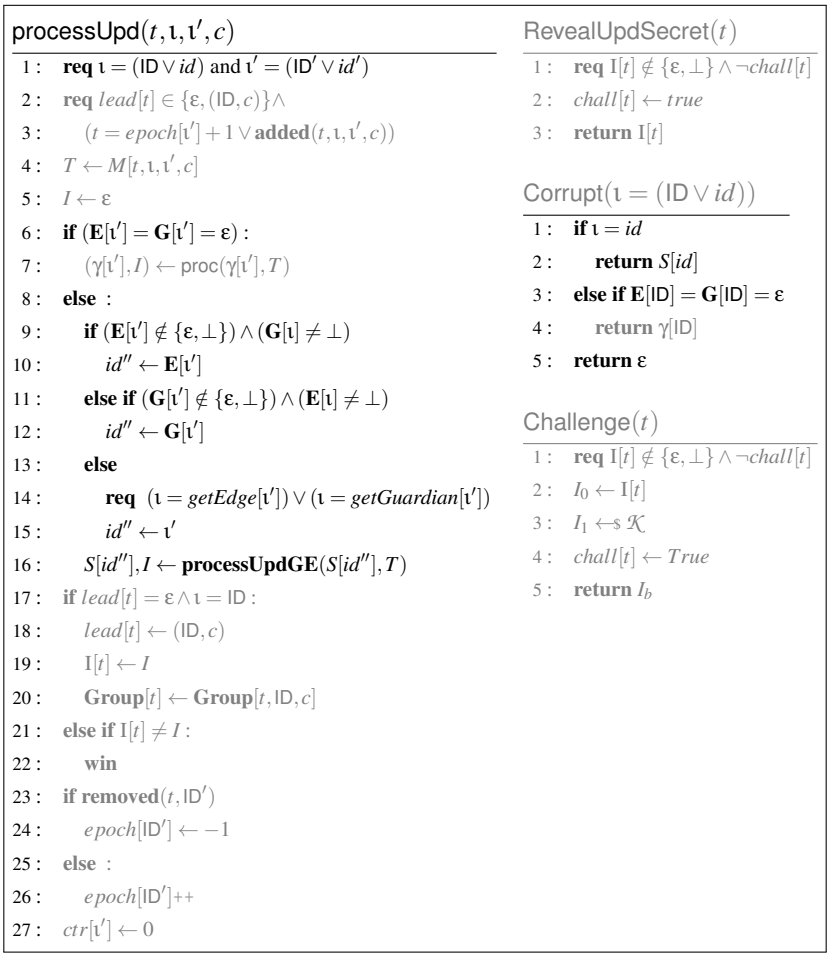


Figure 3: GCGKA experiment continued.

for $\iota = \mathsf{ID} \vee id$, let

$Q_{GPCS}(\iota) = \{$GCGKA-GuardianUpdate$(getGuardian[\iota])\}$

$Q_{FS,PCS}(\iota) = \{$GCGKA-EdgeUpdate(ι), GCGKA-Remove$(*, anchor[\iota])$, GCGKA-Remove$(*,\iota)$, SendUpdate(ι), GCGKA-ExitGship$(\iota,*)\}$

$Q_{FS,PCS,GPCS}(\iota) = Q_{GPCS} \cup Q_{FS,PCS}$.

(a) The list of experiment queries the different security notions allow.

X-safe$(q_1, \cdots, q_q)$

1 : **for** *(i, j) s.t.* $\mathbf{q}_i =$ Corrupt(ι) *where* $\iota = (\mathsf{ID} \vee id)$ *and* $\mathbf{q}_j =$ **chall**(t^*) *for some* t^*
2 : if q2e$(\mathbf{q}_i) \leq t^*$ *and* $\nexists k$ s.t. $0 <$ q2e$(\mathbf{q}_i) <$ q2e$(\mathbf{q}_k) \leq t^*$ and
3 : $(\mathbf{q}_k \in Q_{\mathbf{X}}) \vee (\iota = id_G \wedge (\mathbf{q}_k \in Q_{GPCS}))$
4 : return 0
5 : if q2e$(\mathbf{q}_i) > t^*$ *and* $\exists k$ s.t. q2e$(\mathbf{q}_k) \leq t^*$ and $\mathbf{q}_k =$ no-del(ι)
6 : return 0
7 : return 1

(b) Safety predicate, where $\mathbf{X} \in \{[FS, PCS], [FS, PCS, GPCS]\}$.

Figure 4: **X-safe** adapted from [7], prevents trivial attacks by an attacker to compute the update secret in a challenge epoch t^* using the state of a party ι in some epoch t and the control messages observed on the network. This occurs when an ι stops updating before epoch t^* and is corrupted afterwards or when ι has not performed an update or is removed before epoch t^*. The function q2e$(\mathbf{q})$ returns the epoch corresponding to a query $\mathbf{q}$.

are stored in **M**[]. Like in CGKA, **Group**[] tracks the underlying CGKA group members (including anchors) while **E**[] and **G**[] specifically keep track of edge and guardian members, respectively.

Creation and Maintenance. To create a edge-guardian pair, first a call to createGroup must be made to establish the underlying group. Then, adding, removing, and updating can be accomplished through specific queries. For correctness, checks are made to ensure only valid members of **Group**[] can be added as edges, duplicate adds and removes cannot occur, and only edges can add or remove valid guardians.

Committing Updates. Similar to CGKA, all group operations are published to **M**[] via a control message T which is accessible to the adversary in a read-only mode and is used by the server to adjudicate the order of commits. **M**[] is indexed on the epoch of the intended operation, the ID of the caller, the ID of the affected member, and the counter value. Since control messages can, in some constructions, be between the edge and guardian as non-CGKA control messages, we denote $T_{CGKA} \subseteq \mathbf{M}[]$ as the subset that is specific to CGKA. (See [7, Section 3.2] for how the *send-update*() and *deliver*() within CGKA are used to deconflict and commit updates atomically.)

Reveal and Corrupt. Reveal follows that of CGKA functionality, while corruption is allowed for guardians, edges, and regular group members. Anchors associated to edges or guardians are specifically disallowed – these are logical constructs held in memory by the edge/guardian, so we follow real-world use where a corruption to access such state would be via the edge or guardian. To examine the security properties of the GCGKA compared too CGKA, we need to fine tune adversarial reveal and corruption capabilities. This is done through modifying the safety predicate from CGKA, which captures FS and PCS, to also capture GPCS via guardian-edge pairs (Figure 4b).

Definition 7 (GCGKA Security)**.** *Let GCGKA be an extension of CGKA as defined in Definition 6, let* Π *be a GCGKA protocol and let the adversary* $\mathcal{A}$ *be a PPT algorithm against* Π *as defined in Figures 2, 3 and 4b. We define the adversarial advantage of a* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{X-GCGKA}_{\Pi-\mathcal{A}}(\lambda) = \left|\Pr\left[Exp^{X-GCGKA}_{\Pi-\mathcal{A}}(\lambda)\right] - \frac{1}{2}\right|$$

and say that the protocol Π *is* $\mathbf{X}-$GCGKA-secure *if* $\mathsf{Adv}^{GCGKA}_{\Pi-\mathcal{A}}(\lambda)$ *is negligible for all* $\mathcal{A}$.

Win. The adversary wins the GCGKA experiment by correctly answering if the presented update key, is real or random. The safety predicate enforces PCS, FS, and GPCS in the win condition and denies trivial wins by returning 0 to the adversary. As in CGKA, the function q2e(**q**) returns the epoch corresponding to the query **q**. This epoch is relative to the ι, which may be an anchor ID or guardian/edge *id*, used as input to the query **q**. For the two trivial wins, an adversary a) queries Corrupt for an ι in a epoch and issues a challenge against its update secret before ι or it's guardian makes an update; or b) disabled key deletions for ι at some point before the Corrupt query was issued for ι.

6 Protocol Constructions

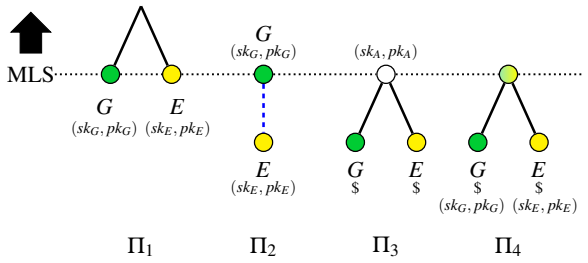
The breadth of GCGKA variations can be seen through three binary design decision points: whether to architect the edge-guardian pair organically within a protocol or extended off of it, whether the edge and guardian devices share an signing key, and whether the edge and guardian devices share randomness. These decisions yield the possible permutations of guardian-edge pair designs shown Figure 5b.

We choose MLS as the CGKA basis for our protocol designs, and Signal [3] as a CKA example where required. For simplicity, we use CGKA and CKA functions to the greatest extent possible. This also allows for clearer extrapolation to other constructions. For the mapping of other and underlying CGKA algorithms (init, add, rem, upd, proc) to MLS, see [7]; we assume these mappings and present the additional GCGKA functions relying on them.

Of the eight possible combinations in Figure 5b, three are dismissed due to incompatibility with secure group protocol notions. In (a), having a guardian edge pair anchored on distinct nodes inside the MLS protocol with unique signing keys but shared randomness removes all possibility of achieving PCS: if an attacker compromises one node, it would be able to compute any new keys that the other uses to heal the group with. This would nullify the goal of maintaining the security of the underlying CGKA. Next, consider (b) and (c); having distinct anchors and shared signing keys: this combination implies clones being allowed as group members, which invalidates non-repudiation and the MLS invariant of unique identities. In contrast, we do consider cases where two different signature keys are registered at the same anchor. The rest of this section explores the implications of our selection choice. Figure 5a gives a conceptual overview, while Figure 6 together with Figure 7 specifies the protocol description.

Π_1 – MLS siblings Π_1 is nearly a direct copy of MLS with edge and guardian as distinct members. Recall that MLS manages keys in a logical binary tree where the group members are represented as leaf nodes and the shared session key as the root. MLS has been shown to achieve FS and PCS for its group members [12], but if a user is unable to update (i.e., is in limited mode) the security of the entire group is affected as it leaves a path from a leaf node to root unaltered [13].

This construction requires guardian and edge device to be siblings in the MLS tree so that they share one path to the root (see Figure 6). This way, when the edge enters limited mode guardian-updates will directly affect their parent node and the shared path to the root, meaning that such keys do not



(a) Guardian GCGKA Π constructions in the context of MLS. Π_5 and Π_6 are described below.

Shared Anchor	**Shared** (sk, pk)	**Shared** \$	Π
0	0	0	1
0	0	1	(a)
0	1	0	(b)
0	1	1	(c)
1	0	0	2 / 6
1	0	1	4a/b
1	1	0	5
1	1	1	3a/b

(b) Truth table of design choices.

Figure 5: GCGKA protocol options. Π_3 and Π_4 are separated into a/b cases depending on if the shared randomness is kept in active memory (e.g., part of $\gamma[id_E]$) or pre-installed in a secure module as may be done with signature keys. Options 5 and 6 mirror 3 and 4 except with distinct random tapes (this necessitates an external channel between G and E for both 5 and 6). An anchor refers to the underlying CGKA protocol's group member's logical node. A signature, (sk, pk), refers to the signing keys belonging to that node. Randomness, \$, refers to the seed used in key generation (see Remark 2).

stay stagnant. Generally, this option doesn't achieve GPCS since only edge removal or self-update can heal the group (i.e. CGKA PCS). While the sibling construct provides a freshness benefit, the main security goals we consider in this option (FS, PCS, GPCS) apply equally to a case where the edge and guardian nodes are not siblings in the MLS tree. Use of updateable public key encryption (UPKE) could improve forward secrecy [23], but this is outside the scope of this work.

Π_2 – Message Forwarding In Π_2, the guardian is an MLS group member, with a separate forwarding channel maintained between the edge and guardian. We use a CKA channel for simplicity in the shown construction. A notable consideration for Π_2 is that not only is the guardian effectively set to man-in-the-middle the connection between the edge and the group, but it is impossible for the edge to remove the guardian to regain underlying group control. Observe, **exitGship** relies on edge action to terminate the CKA channel with the guardian. This is an inherent issue to Π_2– any direct link between the edge and main group would change the protocol to function like e.g., Π_6.

Π_3 – Shared Signature keys, Shared Rand. Both the guardian and the edge share control of one anchor in this variant. Furthermore, we allow the edge to be aware of the guardian's internal choices through having shared randomness. We split Π_3 into two cases with separate constructions based on how the randomness is shared: Π_{3a} for a copy into active memory and Π_{3b} for a copy placed in secure hardware.

Remark 2. *If true random number generators are used separately in edge and guardian devices, a shared random tape is not possible. Therefore, instead a seed is shared between edge and guardian. The seed is updated in a deterministic manner. Without the knowledge of the seed, key generation will look random, while access to the seed on either device implies compromise of both.*

Π_{3a} – Unprotected Shared Randomness In this subvariant, the shared randomness is copied into memory that is accessible to the adversary under a $\mathsf{Corrupt}(\mathsf{ID})$ query. Once an adversary corrupts the edge or guardian, both are corrupted since the randomness is a direct duplication. Furthermore, the randomness, now in adversarial possession, is the same randomness used to generate new keys, the group cannot obtain PCS. PCS can only be achieved through removal of both the edge and guardian. GPCS can never be attained.

Π_{3b} – Protected Shared Randomness In this subvariant, the shared randomness is copied into a secure hardware component and is kept separate from the active state. This is analogous to the protections placed on signature keys. Such memory is not accessible to the adversary under a $\mathsf{Corrupt}(\mathsf{ID})$ query. If an adversary were able to corrupt the state of the edge or guardian, they do not get access to the randomness. Thus, PCS is achievable through a normal update process. Furthermore, GPCS is achieved since the guardian can issue an update to the rest of the group that the corrupted edge can generate locally. Since the update secret keying material does not need to be sent to the edge, an adversary is unable to gain access to the new group key even though they have a copy of the local state $S[id_E]$. Thus, through automated guardian key updates, PCS can be achieved even if the corrupted edge is in limited mode.

Π_4 – Distinct Signature Keys, Shared Randomness Instead of allowing end users in the underlying protocol to register only a single authentication key pair, we allow multiple signing keys for a single CGKA node. By having the guardian and the edge device possess two distinct but registered signing keys the edge-guardian impersonation behaviour seen in Π_3 is impossible. However, this does require that CGKA members (in this case, MLS members), can associate more than one signing key to a given leaf member node.

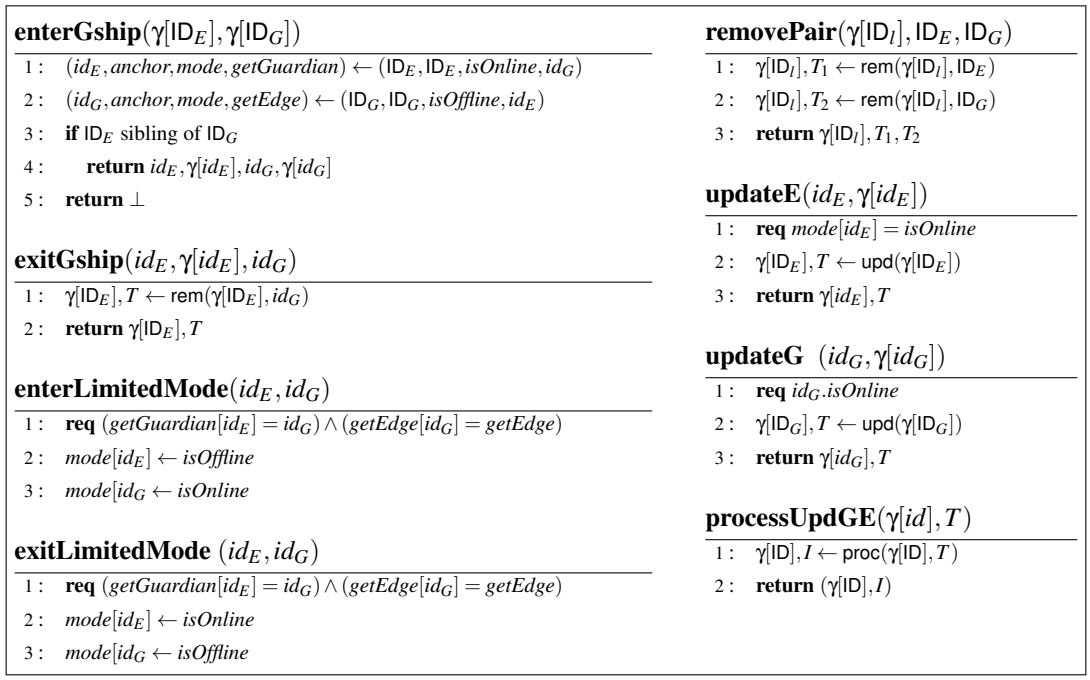


Figure 6: Construction of the additional GCGKA Π_1 algorithms, including algorithms for entering and exiting guardianship mode (**enterGship),exitGship**), entering and exiting limited mode (**enterLimitedMode, exitLimitedMode**), removing an edge-guardian pair (**removePair**), updating an edge or guardian (**updateE, updateG**), and processing a received update (**processUpdGE**).

By expanding the edge and guardian id to include the new signing keys the remaining protocol is identical to that presented in Figure 7 Π_{3a} and Π_{3b}. The new edge id and guardian with anchor ID will now be $id_E = (A, \mathsf{ID}, \cdot, \mathsf{sk}_A)$ and $id_G = (B, \mathsf{ID}, \cdot, \mathsf{sk}_B)$. The discussion about security levels from the previous sections remain.

Π_5 – Shared Anchor and Signature key, Distinct Rand. In Π_5, a guardian and edge share an anchor and signature key pair, as in Π_3. Unlike Π_3, however, the guardian and edge retain distinct individual states and randomness. This necessitates a separate, non-CGKA channel to be maintained between them, similar to Π_2, to share update information.

This option limits the fallout risk from edge compromise as an adversary does not also automatically obtain the guardian state. Separation in to unprotected and protected randomness is unnecessary since distinct random tapes mean that both devices can use random number generators. GPCS is impossible, however, for the same reason. PCS and FS are maintained. If a signature key is compromised, then both the edge and guardian can be impersonated. Refer to Figure 7 for the construction.

Π_6 – Shared Anchor, Distinct Signature Key and Rand. Π_6 mirrors Π_4 except that distinct random tapes are used. As in Π_5, this solves issues relating to secure storage of a shared random tape, but also necessitates a separate channel between G and E. In our construction, this is a CKA channel.

7 Analysis

Given the above insights we can summarize overall trade-offs among the proposed options. The first step is to show that all the protocols are GCGKA protocols, i.e. they behave according to Definition 6 and fulfill the four mandatory requirements: correctness, privacy, FS, PCS.

Theorem 1. *[FS and PCS Security] Consider Π_i, $i \in \{1, 2, 3b, 4b, 5, 6\}$. Then Π_i is [FS,PCS]-GCGKA-secure secure under the CGKA security of the CGKA and the CKA security of the CKA. That is, for any PPT algorithm $\mathcal{A}$ against the [FS,PCS]-GCGKA security experiment, $\mathsf{Adv}^{[\mathrm{FS,PCS}]-GCGKA}_{\Pi-\mathcal{A}}(\lambda)$ is negligible.*

Theorem 2. *[GPCS Security] The protocols Π_{3b} and Π_{4b} are [FS,PCS,GPCS]-GCGKA-secure under the CGKA security of the CGKA and the CKA security of the CKA. That is, for any PPT algorithm $\mathcal{A}$ against the [FS,PCS,GPCS]-GCGKA security experiment, $\mathsf{Adv}^{[\mathrm{FS,PCS,GPCS}]-GCGKA}_{\Pi-\mathcal{A}}(\lambda)$ is negligible.*

Due to space constraints, the proofs for Theorem 1 and Theorem 2 can be found in Appendix A.

enterGship$(\gamma[\mathsf{ID}],\gamma[\mathsf{ID}])$

$\mathsf{k} \leftarrow\$ \mathcal{K}$
if $\Pi = \Pi_2 \vee \Pi_5 \vee \Pi_6$
  $\gamma^A \leftarrow$ CKA-Init-A(k)
  $\gamma^B \leftarrow$ CKA-Init-B(k)
else
  $\gamma^A, \gamma^B \leftarrow \varepsilon$
$(id_E, anchor, mode, getGuardian) \leftarrow (A, \mathsf{ID}, isOnline, id_G)$
$(id_G, anchor, mode, getEdge) \leftarrow (B, \mathsf{ID}, isOffline, id_E)$
if $\Pi = \Pi_2$
  $S[id_E] \leftarrow ((\gamma^A, \mathsf{k}), \varepsilon)$
else
  $S[id_E] \leftarrow ((\gamma^A, \mathsf{k}), \gamma[\mathsf{ID}])$
$S[id_G] \leftarrow ((\gamma^B, \mathsf{k}), \gamma[\mathsf{ID}])$
return $id_E, (\gamma^A, \mathsf{k}), id_G, (\gamma^B, \mathsf{k})$

exitGship$(id_E, S[id_E], id_G)$

$(\gamma[id_E], \gamma[\mathsf{ID}_E]) \leftarrow S[id_E]$
req $getGuardian[id_E] = id_G$
req $getEdge[id_G] = id_E$
req $anchor[id_G] = anchor[id_E]$
if $\Pi = \Pi_2$
  $\gamma[\mathsf{ID}_E], T \leftarrow (\perp, \perp)$
else
  $\gamma[\mathsf{ID}_E], T \leftarrow \mathsf{rem}(\gamma[\mathsf{ID}_E], \mathsf{ID}_E)$
$S[id_E] \leftarrow (\gamma[id_E], \gamma[\mathsf{ID}_E])$
return $\gamma[\mathsf{ID}_E], T$

enterLimitedMode(id_E, id_G)

req $(getGuardian[id_E] = id_G) \wedge (getEdge[id_G] = getEdge)$
$mode[id_E] \leftarrow isOffline$
$mode[id_G] \leftarrow isOnline$

exitLimitedMode (id_E, id_G)

req $(getGuardian[id_E] = id_G) \wedge (getEdge[id_G] = getEdge)$
$mode[id_E] \leftarrow isOnline$
$mode[id_G] \leftarrow isOffline$

removePair$(\gamma[\mathsf{ID}_l], \mathsf{ID}, \mathsf{ID})$

$\gamma[\mathsf{ID}_l], T_1 \leftarrow \mathsf{rem}(\gamma[\mathsf{ID}_l], \mathsf{ID})$
return $\gamma[\mathsf{ID}], T_1, \varepsilon$

updateE$(id_E, S[id_E])$

req $mode[id_E] = isOnline$
$((\gamma, \mathsf{k}), \gamma[\mathsf{ID}]) \leftarrow S[id_E]$
if $\Pi \neq \Pi_2$
  $(\gamma[\mathsf{ID}], T_2) \leftarrow \mathsf{upd}(\gamma[\mathsf{ID}]; \mathsf{k})$
if $\Pi = \Pi_3 \vee \Pi_4$
  $T_1 \leftarrow KDF(\mathsf{k}, \mathtt{cntrl})$
  $\mathsf{k} \leftarrow KDF(\mathsf{k})$
else if $\Pi = \Pi_2 \vee \Pi_5 \vee \Pi_6$
  $(\gamma, T_1, \mathsf{k}) \leftarrow\$$ CKA-S(γ)
if $\Pi = \Pi_2$
  $S[id_E] \leftarrow ((\gamma, \mathsf{k}), \varepsilon),\ T_2 \leftarrow \varepsilon$
else
  $S[id_E] \leftarrow ((\gamma, \mathsf{k}), \gamma[\mathsf{ID}])$
return $S[id_E], (T_1, T_2)$

updateG$(id_G, S[id_G])$

req $mode[id_G] = isOnline$
$((\gamma, \mathsf{k}), \gamma[\mathsf{ID}]) \leftarrow S[id_G]$
$(\gamma[\mathsf{ID}], T_2) \leftarrow \mathsf{upd}(\gamma[\mathsf{ID}]; \mathsf{k})$
if $\Pi = \Pi_3 \vee \Pi_4$
  $T_1 \leftarrow KDF(\mathsf{k}, \mathtt{cntrl})$
  $\mathsf{k} \leftarrow KDF(\mathsf{k})$
else if $\Pi = \Pi_2 \vee \Pi_5 \vee \Pi_6$
  $(\gamma, T_1, \mathsf{k}) \leftarrow\$$ CKA-S(γ)
$S[id_G] \leftarrow ((\gamma, \mathsf{k}), \gamma[\mathsf{ID}])$
return $(S[id_G], (T_1, T_2)$

processUpdGE$(S[id], T)$

$((\gamma, \mathsf{k}), \gamma[\mathsf{ID}]) \leftarrow S[id]$
$(T_1, T_2) \leftarrow T$
if $\Pi = \Pi_3 \vee \Pi_4$
  req $T_1 = KDF(\mathsf{k}, \mathtt{cntrl})$
if $\Pi = \Pi_3 \vee \Pi_4 \vee \Pi_5 \vee \Pi_6$
  $\gamma[\mathsf{ID}], I \leftarrow \mathsf{proc}(\gamma[\mathsf{ID}], T_2; \mathsf{k})$
if $\Pi = \Pi_3 \vee \Pi_4$
  $\mathsf{k} \leftarrow KDF(\mathsf{k})$
  $S[id] \leftarrow ((\gamma, \mathsf{k}), \gamma[\mathsf{ID}])$
else if $\Pi = (\Pi_5 \vee \Pi_6) \vee (\Pi = \Pi_2 \wedge (id = id_G))$
  $(\gamma, \mathsf{k}) \leftarrow$ CKA-R(γ, T_1)
  $S[id] \leftarrow ((\gamma, \mathsf{k}), \gamma[\mathsf{ID}])$
else if $\Pi = \Pi_2 \wedge (id = id_E)$
  $(\gamma, \mathsf{k}) \leftarrow$ CKA-R(γ, T_1)
  $S[id] \leftarrow ((\gamma, \mathsf{k}), \varepsilon),\ I \leftarrow \varepsilon$
return $(S[id], I)$

Figure 7: Constructions for the additional GCGKA Π_2, Π_3 and Π_5 algorithms. Π_{3a} and Π_{3b} are differentiated in the storage protection of k. Furthermore, this construction covers Π_4 and Π_6 as Π_3 and Π_5 resp., when id expanded to $id_E = (A, \mathsf{ID}, \cdot, \mathsf{sk}_A)$ and $id_G = (B, \mathsf{ID}, \cdot, \mathsf{sk}_B)$ for edge and guardian.

Theorem 1 forms a baseline check that the guardianship introduction has not affected the underlying security of the CGKA for the edge. As noted, Π_{3a} and Π_{4a} do not pass this check, while in Π_2 achieves FS and PCS only to the extent of the CKA channel between the edge and guardian – the edge is completely outside the FS and PCS scope of the group protocol. Theorem 2 meanwhile demonstrates the viability of having the guardian issue updates on behalf of the edge. To achieve GPCS, a core assumption is that the shared randomness can be installed in protected hardware in the same way that long-term signing keys are, and are therefore out of scope of a Corrupt query. Thus, while GPCS is fundamentally a cryptographic notion, achieving it relies on various architectural decisions.

8 Architectural comparison

Security considerations fall into two categories: protocol guarantees which can be analyzed using provable security, and properties that are affected based on architectural choices. Architectural questions extend beyond security and into reliability and usability within the use environment constraints. The DS used for message transport may further influence the properties that are achievable. A summary of the architectural-related properties associated with each protocol is shown in Table 1. Their associated overhead costs are shown in Table 2.

Connectivity: In the edge-guardian scenario, it is assumed that the edge may be in limited/receive-only mode for extended periods of time. During that time, it is expected that the guardian can provide updates – this, however, raises the question of whether the guardian *must* be online as a functionality requirement.

With the exception of Π_2 none of the proposed protocols require both an online guardian and edge device to function. In the case of Π_2, the edge is entirely dependent on the reliable connectivity of the guardian. If, e.g., the user is traveling with the edge device and left the guardian in a secure location at home, this could come with network delays as the communication must route through the guardian. Also in Π_2 it is impossible for the edge device to participate in the group independently of the guardian even if online.

Guardian Removal: Device removal has differing effects across the protocol options, meaning that **exitGship** results in differing behaviors. In Π_1, a guardian can be removed from the group without affecting its edge. The same holds for Π_4 and Π_6. In contrast, removal of the guardian in Π_2, Π_3, or Π_5 will necessarily incur the removal of the edge. For Π_2 the guardian is the gateway to the group for the edge while in Π_3 and Π_5 the guardian and edge share a signing key – thus removal of the signing key identity automatically incurs removal of the edge.

If an attempt is made in Π_{3a}, Π_{3b}, or Π_5 to remove a guardian while maintaining the edge's access to the group, the guardian will still be able to decrypt messages in the case of either Π_{3a} or Π_{3b}, and can impersonate an update and regain access while locking out the edge in the case of Π_5.

Traceability: In some protocols, other group members may be aware of the guardianship and can trace the online/offline status of the guardian or limited mode for the edge device. In other protocols, the DS must have knowledge.

In the case of Π_1, Π_4 and Π_6 the edge and guardian have separate signature keys used when preforming an update into the MLS group. As such, any group member will be able to differentiate between edge and guardian, and thus determine the operational mode in play.

In the remaining cases signature keys are identical for both guardian and edge. Thus an arbitrary group member will not be able to determine which of the two entities is online.

The *distribution service (DS)*, however, being used to deliver data between members in the correct order, could trace the edge/guardian relationship and online/offline status – if a single DS is used. Under Π_2 it is possible to implement the channel between the edge and guardian using a separate DS from the CGKA, as no coordination with the CGKA updates is necessary. In the case of Π_3 and Π_5, the DS must have awareness of the edge/guardian pairing in order to deliver messages. It may also be aware of online/offline statuses. This is dependent on a push or pull design of the DS. The DS must be able to distinguish between the edge and guardian in order to send them both messages in a push context, whereas the edge-guardian relationship can be more obscured to the DS in a pull context.

As a core group member, it is not a requirement that the DS is aware of the guardian existence or status; however, a lack of awareness implies that the guardian is privy to all application messages. In other words, either the DS delivers messages to the guardian based on activity status, or the DS always delivers messages to the guardian.

Extensibility: A natural question on guardianship is the extensibility to place multiple edge devices with a single guardian. While we do not model this under the GCGKA experiment, it is not unreasonable to consider the option. Π_1 loses the sibling connection if further edges are added. Π_2 can easily be extended. Likewise, Π_3, Π_4, Π_5, and Π_6 can have further edges added to the anchor with a single guardian.

There are security costs to some such arrangements – in Π_3 and Π_4, for example, the same random tape would necessarily be shared among all such edge devices as well as the guardian, creating more opportunity for compromise. In the case of Π_{3a}, Π_{4a}, where the shared randomness is kept in active memory, this is especially unsafe. Even in Π_{3b} and Π_{4b}, where the shared randomness is kept in protective memory, any one edge is still at risk from compromise of another as the same assumption do not apply to edges as guardians (i.e., that the guardian is kept in a secure location). In contrast, Π_1 does not present such hazards among edge devices.

Impersonation Risk: There are a number of straightforward observations that can be made on the comparative

Feature	Π_1	Π_2	Π_{3a}	Π_{3b}	Π_{4a}	Π_{4b}	Π_5	Π_6
Extensible to multiple edges	○	●	●	●	●	●	●	●
Guardian can be offline	●	○	●	●	●	●	●	●
Guardian removal without edge	●	○	○	○	●	●	○	●
Traceability of guardianship	●	○	◐	◐	●	●	◐	●
Low impersonation risk	●	○	○	○	●	●	○	●
FS	●	◐	●	●	●	●	●	●
PCS	●	◐	○	●	○	●	●	●
GPCS	○	○	○	●	○	●	○	○

Table 1: Architectural feature comparison. Green circles mean "Supported", Unfilled circles mean "Unsupported", and half-colored circles mean "It Depends". Depending on use case, different features may be desirable.

Notable Changes	Π_1	Π_2	Π_{3a}	Π_{3b}	Π_{4a}	Π_{4b}	Π_5	Π_6
Increased Tree Size	✗							
Preloaded, Secure Storage of Shared Random Seed			✗	✗	✗	✗		
Additional Channel Maintenance		✗					✗	✗
Additional Per-Device Signature Key in MLS					✗	✗		✗

Table 2: Summary of changes from baseline MLS across various constructions.

risk of a guardian impersonating an edge device. These are in part dependent on whether application messages are signed or only group control messages – a choice dependent on the underlying CGKE, e.g., MLS or other.

In Π_1, Π_{4a}, Π_{4b}, and Π_6, the edge device and guardian device will have distinct signing keys and will not have any knowledge of the others keying material. This assures non-repudiation. Impersonation detection by other group members, as well as by the edge and guardian device may therefore be possible. This is true for both tree updates as well as regular message transmission.

For Π_2, Π_{3a}, Π_{3b}, and Π_5, the guardian possesses a copy of the edge's signing key or acts as the anchor relay, and can therefore impersonate the edge device – in fact, impersonation is a requirement for normal functionality. This means that guardianship is opaque to other group members but comes at the cost of non-repudiation (for Π_{3a}, Π_{3b}, and Π_5) as well as an increased reliance on the honesty/security of the guardian.

Costs: Each protocol presented has some additional cost pertaining to storage, computation, or additional hardware. Table 2 gives an overview of notable changes based on each protocol. Further calculation can show costs for specific cases. These should be taken into account when compared to possible improvements, as seen from Table 1.

With the exception of Π_1, all the other proposed protocols use the same anchor for both guardian and edge device. This means that in order for Π_1 to function, an additional MLS leaf node needs to be added per edge device; resulting in a greater tree of depth. Since the depth of the MLS tree directly correlates to the number of encryptions and decryptions needed to be preformed for each update, Π_1 would result in one encryption/decryption per update.

For protocols Π_4 and Π_6 the tree size does not need to be altered but an extension change to MLS is required. Specifically, the allowance of multiple authentication keys to a singular MLS member is required. Π_4 additionally needs extra key management to load the shared seed, a requirement also found in Π_3. In Π_{3b} and Π_{4b}, secure hardware requirements also needs to be met. While need for shared randomness is not found in Π_6, it still requires additional key management in order to keep a separate communication channel between edge and guardian. This similarly applies to Π_2 and Π_5.

9 Conclusion

While prior work has considered PCS to be uniquely tied to the ability for a compromised node to issue key updates, we have shown that it is possible for a designated third party to perform this function on the behalf of vulnerable members through GPCS. Achieving GPCS is non-trivial. Both it and the overall security of the protocol are highly dependent on a number of system architectural choices, and such decisions can introduce trade-offs.

As shown in Section 7, among the design space options, only Π_{3b} and Π_{4b} achieve GPCS security. Given the options listed, Π_{4b} nominally demonstrates the best selection of properties. However, given the full discussion provided in this paper, one can note that this is context, security goal, and environment dependent.

References

[1] Martin R. Albrecht, Sofia Celi, Benjamin Dowling, and Daniel Jones. Practically-exploitable cryptographic vulnerabilities in matrix. Cryptology ePrint Archive, Paper 2023/485, 2023. https://eprint.iacr.org/2023/485.

[2] Martin R. Albrecht, Lenka Mareková, Kenneth G. Paterson, and Igors Stepanovs. Four attacks and a proof for telegram. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 87–106, 2022.

[3] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 129–158, Cham, 2019. Springer International Publishing.

[4] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the ietf mls standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 248–277, Cham, 2020. Springer International Publishing.

[5] Joël Alwen, Daniel Jost, and Marta Mularczyk. On the insider security of mls. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 34–68, Cham, 2022. Springer Nature Switzerland.

[6] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the signal protocol. Cryptology ePrint Archive, Paper 2018/1037, 2018. https://eprint.iacr.org/2018/1037.

[7] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. *Security Analysis and Improvements for the IETF MLS Standard for Group Messaging*, pages 248–277. 08 2020.

[8] Alwen, Joël and Sandro Coretti and Yevgeniy Dodis and Yiannis 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*, CCS '21, page 1463–1483, 2021.

[9] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. RFC 9420, July 2023.

[10] Richard Barnes, Jon Millican, Emad Omara, Katriel Cohn-Gordon, and Raphael Robert. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-20, IETF Secretariat, May 2023. https://datatracker.ietf.org/doc/draft-ietf-mls-protocol/.

[11] Corina-Elena Bogos, Răzvan Mocanu, and Emil Simion. A security analysis comparison between signal, whatsapp and telegram. Cryptology ePrint Archive, Paper 2023/071, 2023. https://eprint.iacr.org/2023/071.

[12] Christina Brzuska, Eric Cornelissen, and Konrad Kohbrok. Cryptographic security of the mls rfc, draft 11. *IACR Cryptol. ePrint Arch.*, 2021:137, 2021.

[13] Konrad Kohbrok Cas Cremers, Britta Hale. The complexities of healing in secure group messaging: Why cross-group effects matter. *USENIX*, pages 1847–1864, 2021.

[14] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy, (Euro S&P)*, pages 451–466, 2017.

[15] Benjamin Dowling and Britta Hale. Secure Messaging Authentication against Active Man-in-the-Middle Attacks. In *2021 IEEE European Symposium on Security and Privacy, (Euro S&P)*, 2021.

[16] Benjamin Dowling and Britta Hale. Authenticated continuous key agreement: Active mitm detection and prevention. Cryptology ePrint Archive, Paper 2023/228, 2023. https://eprint.iacr.org/2023/228.

[17] F. Betül Durak and Serge Vaudenay. Bidirectional asynchronous ratcheted key agreement without key-update primitives. *IACR Cryptol. ePrint Arch.*, 2018:889, 2018.

[18] S. Frankel and S. Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. https://www.rfc-editor.org/rfc/rfc6071, February 2011.

[19] Britta Hale and Chelsea Komlo. On end-to-end encryption. Cryptology ePrint Archive, Paper 2022/449, 2022. https://eprint.iacr.org/2022/449.

[20] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport, RFC 9000. https://datatracker.ietf.org/doc/rfc9000/, February 2022.

[21] Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure

messaging. In *Advances in Cryptology - EUROCRYPT 2019*, volume 11476 of *Lecture Notes in Computer Science*, pages 159–188. Springer, 2019.

[22] Daniel Jost, Ueli Maurer, and Marta Mularczyk. A unified and composable take on ratcheting. *IACR Cryptol. ePrint Arch.*, 2019:694, 2019.

[23] Konrad Kohbrok. Subject: [mls] improve fs granularity at a cost. MLS Mailing List, Jan 2019.

[24] Hugo Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 631–648. Springer Berlin Heidelberg, 2010.

[25] Ian Levy and Crispin Robinson. Principles for a more informed exceptional access debate. Lawfare, November 2019.

[26] Moxie Marlinspike. Advanced cryptographic ratcheting. https://signal.org/blog/advanced-ratcheting/, November 2013.

[27] Moxie Marlinspike. Facebook Messenger deploys Signal Protocol for end-to-end encryption. 2016.

[28] Moxie Marlinspike and Trevor Perrin. The Signal Protocol. Technical report, Signal.org, November 2016.

[29] Sam Meredith. Apple, Google and WhatsApp condemn UK proposal to eavesdrop on encrypted messages. CNBC, May 2019.

[30] Bertram Poettering and P. Rösler. Asynchronous Ratcheted Key Exchange. In *CRYPTO, '18*, 2018.

[31] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. https://tools.ietf.org/html/rfc8446, August 2018.

[32] 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, (Euro S&P)*, pages 415–429, 2018.

A GCGKA Analysis

Recall the following theorems.

Theorem 1. *[FS and PCS Security] Consider* Π_i, $i \in \{1,2,3b,4b,5,6\}$. *Then* Π_i *is [FS,PCS]-GCGKA-secure secure under the CGKA security of the CGKA and the CKA security of the CKA. That is, for any PPT algorithm* $\mathcal{A}$ *against the [FS,PCS]-GCGKA security experiment,* $\mathsf{Adv}^{[\mathrm{FS,PCS}]-\mathit{GCGKA}}_{\Pi-\mathcal{A}}(\lambda)$ *is negligible.*

Theorem 2. *[GPCS Security] The protocols* Π_{3b} *and* Π_{4b} *are [FS,PCS,GPCS]-GCGKA-secure under the CGKA security of the CGKA and the CKA security of the CKA. That is, for any PPT algorithm* $\mathcal{A}$ *against the [FS,PCS,GPCS]-GCGKA security experiment,* $\mathsf{Adv}^{[\mathrm{FS,PCS,GPCS}]-\mathit{GCGKA}}_{\Pi-\mathcal{A}}(\lambda)$ *is negligible.*

We provide sketch proofs for each of the different protocols.

A.1 Security Analysis of Π_1

Proof Theorem 1. Since Π_1 builds directly on CGKA protocol CGKA it follows directly from Definition 6 and Figure 6 that Π_1 is a GCGKA protocol.

To show that Π_1 is $[FS,PCS]-$GCGKA secure we need to show that any adversary against Π_1 has negligible advantage.

Let $\mathcal{A}$ be a PPT algorithm against Π_1, Figure 6, with advantage $\mathsf{Adv}^{[\mathrm{FS,PCS}]-\mathsf{GCGKA}}_{\Pi_1-\mathcal{A}}(\lambda)$. We use $\mathcal{A}$ to create an adversary $\mathcal{B}_1$, Figure 8, against CGKA protocol CGKA with advantage $\mathsf{Adv}^{\mathsf{CGKA}}_{\mathrm{CGKA}-\mathcal{B}_1}(\lambda) \geq \mathsf{Adv}^{[\mathrm{FS,PCS}]-\mathsf{GCGKA}}_{\Pi_1-\mathcal{A}}(\lambda)$. We use MLS as CGKA, and since MLS has been shown to be CGKA secure [7] meaning that the advantage of $\mathcal{A}$ must be negligible, and by extension that Π_1 is $[FS,PCS]-$GCGKA secure.

$\mathcal{B}_1$ against $\mathrm{Exp}^{\mathsf{CGKA}}_{\mathrm{CGKA}}$.

Run an internal version of $\mathcal{A}$.
When $\mathcal{A}$ outputs b output b.
if $\mathcal{A}$ sends a GCGKA-EnterGship($\mathsf{ID}_E, \mathsf{ID}_G$) query:
  if $\mathsf{ID}_E, \mathsf{ID}_G$ are group members.
    Record pair ($\mathsf{ID}_E, \mathsf{ID}_G$, true).
if $\mathcal{A}$ sends a GCGKA-ExitGship(id_E, id_G) query:
  if ($id_E, id_G, \cdot$) is recorded.
    Send a CGKA *remove*(id_E, id_G) to $\mathrm{Exp}^{\mathsf{CGKA}}_{\mathrm{CGKA}}$.
if $\mathcal{A}$ sends a GCGKA-ToggleLimitedMode(id_E) query:
  if (id_E, ID, x) is recorded for some id ID and boolean value x.
    Record ($id_E, \mathsf{ID}, \neg x$).
if $\mathcal{A}$ sends a GCGKA-Remove($\mathsf{ID}, \mathsf{ID}'$) query:
  Send *remove*($\mathsf{ID}, \mathsf{ID}'$) to $\mathrm{Exp}^{\mathsf{CGKA}}_{\mathrm{CGKA}}$.
  if ($\mathsf{ID}', \mathsf{ID}''$) or ($\mathsf{ID}'', \mathsf{ID}'$) is recorded for some id ID''.
    Send *remove*($\mathsf{ID}, \mathsf{ID}''$) to $\mathrm{Exp}^{\mathsf{CGKA}}_{\mathrm{CGKA}}$.
    Delete recording.
if $\mathcal{A}$ sends a GCGKA-EdgeUpdate(id_E) query:
  if (id_E, ID, true) is recorded for some id ID.
    Send update(id_E) to $\mathrm{Exp}^{\mathsf{CGKA}}_{\mathrm{CGKA}}$.
if $\mathcal{A}$ sends a GCGKA-GuardianUpdate(id_G) query:
  if (ID, id_G, false) is recorded for some id ID.
    Send update(id_G) to $\mathrm{Exp}^{\mathsf{CGKA}}_{\mathrm{CGKA}}$.
for any other query $\mathcal{A}$ sends.
  Forward the query to $\mathrm{Exp}^{\mathsf{CGKA}}_{\mathrm{CGKA}}$.
Forward any received/updated information returned from $\mathrm{Exp}^{\mathsf{CGKA}}_{\mathrm{CGKA}}$ to $\mathcal{A}$.

Figure 8: Reduction of a $[FS,PCS]-$GCGKA secure adversary against Π_1

When $\mathcal{A}$ issues a challenge, $\mathcal{B}_1$ will query the same challenge to $\mathrm{Exp}^{\mathsf{GCGKA}}_{\Pi_1}$, then pass the session key back to $\mathcal{A}$. Since

$\mathcal{B}_1$ only does book-keeping before passing all the queries made by $\mathcal{A}$ onto $\text{Exp}^{\text{CGKA}}_{\text{CGKA}}$ $\mathcal{A}$ will not be able to distinguish the difference. A successful guess made by $\mathcal{A}$ will, therefore, result in a successful guess in $\text{Exp}^{\text{CGKA}}_{\text{CGKA}}$.

Finally, to ensure that any valid win for $\mathcal{A}$ is a valid win for $\mathcal{B}_1$ we need to make sure that if the CGKA safety predicate returns 1 then GCGKA safety predicate returns 1. Since we have that $X = |FS, PCS|$ this follows directly from definition; [7], Figure 4b. As a result we get;

$$\text{Adv}^{[\text{FS,PCS}]-\text{GCGKA}}_{\Pi_1-\mathcal{A}}(\lambda) \leq \text{Adv}^{\text{CGKA}}_{\text{CGKA}-\mathcal{B}_1}(\lambda) < \text{negl}(\lambda).$$

□

A.2 Security Analysis of Π_2

Proof Theorem 1. Since Π_2 builds directly on CGKA protocol CGKA it follows directly from Definition 6 and Figure 6 that Π_2 is a GCGKA protocol, the fact that an additional CKA communication channel exist does not interfere here.

To show that Π_2 is $[FS, PCS]-$GCGKA secure we need to show that any adversary against Π_2 has negligible advantage. We perform one game hop, Figure 9, using an adversary $\mathcal{B}_1$ and then reduce the problem to an CGKA adversary $\mathcal{B}_2$.

G0 - G1: The game hop between G0 and G1 exchanges the correct generation of CKA-session keys with randomly selected values. Let $\mathcal{B}_1$ be and adversary that can distinguish between G0 and G1. That means $\mathcal{B}_1$ is an adversary that can determine if k is a real or random CKA session key, i.e. $\mathcal{B}_1$ is an adversary against $\text{Exp}^{\text{CKA-PCS},I^*,\Delta_{\text{CKA}}}_{CKA-\mathcal{B}_1}$. Depending on whom $\mathcal{B}_1$ challenges, edge or guardian, the key might be real or random, meaning the two games are identical, thus we get a factor $\frac{1}{2}$:

$$\frac{1}{2}\left|\Pr[G0] - \Pr[G1]\right| \leq \text{Adv}^{\text{CKA-PCS}}_{\text{CKA}-\mathcal{B}_1}(\lambda).$$

G1 - G2: Not relevant; the two games are identical.

For the final step we create an CGKA adversary $\mathcal{B}_2$ against CGKA, modeled as MLS, using $\mathcal{A}$ against Π_2 by running an internal version of G1 and forwarding CGKA queries to $\text{Exp}^{\text{CGKA}}_{\text{CGKA}}$. Similar arguments for the validity of $\mathcal{B}_2$ against G2 as used in the previous section gives us.

$$\Pr[G2] \leq \text{Adv}^{\text{CGKA}}_{\text{CGKA}-\mathcal{B}_2}(\lambda).$$

As a result we get:

$$\text{Adv}^{\text{GCGKA}-|\text{FS,PCS}|}_{\Pi_2-\mathcal{A}}(\lambda) \leq 2\text{Adv}^{\text{CKA-PCS}}_{\text{CKA}-\mathcal{B}_1}(\lambda) + \text{Adv}^{\text{CGKA}}_{\text{CGKA}-\mathcal{B}_2}(\lambda)$$

□

A.3 Security Analysis of Π_3

We will only give the proof for Π_{3b} the proof for Π_{4b} follows exactly with only a different construction of the edge and guardian id.

Proof of Theorem 2. Since Π_{3b} builds directly on CGKA protocol MLS it follows directly from Definition 6 and Figure 6 that Π_{3b} is a GCGKA protocol, the fact that an additional randomness exist and is utilized during run time does not interfere here.

To show that Π_3 is $[FS, PCS, GPCS]-$GCGKA secure we need to show that any adversary against Π_{3b} has negligible advantage. We perform one game hop, Figure 9, using an adversary $\mathcal{B}_1$ against KDF and then reduce the problem to a CGKA adversary $\mathcal{B}_2$.

G0 - G1: Not relevant; the two games are identical.

G1 - G2: The game hop between G1 and G2 exchanges the KDF derivation of key k with true randomness. Let $\mathcal{B}_1$ be and adversary that can distinguish between G1 and G2, i.e. an adversary against real or random (ROR), and as in the previous section only the case of update is changed leaving us with a factor $\frac{1}{2}$.

$$\frac{1}{2}\left|\Pr[G1] - \Pr[G2]\right| \leq \text{Adv}^{\text{ROR}}_{KDF-\mathcal{B}_1}(\lambda).$$

For the final step we create an CGKA adversary $\mathcal{B}_2$ against CGKA using $\mathcal{A}$ against Π_{3b} by running an internal version of G2 and forwarding CGKA queries to $\text{Exp}^{\text{CGKA}}_{\text{CGKA}}$.

To determine the validity of the win we are not able to rely on the argument made in the previous sections. If forward secrecy holds in $\mathcal{A}$ then it must naturally hold for $\mathcal{B}_2$ as old randomness and keys are deleted after session. The problem occurs with PCS and GPCS; if $\mathcal{A}$ reveals the state of an edge or a guardian and the new state and by extension group key is created deterministically from the previous state PCS will not be achievable. This is the case for Π_{3a} (Π_{4a}). However if the randomness used for deriving the next state/key is kept separate and secure, then after one update preformed by the revealed entity will result in fresh randomness, in other words PCS is achieved. The latter scenario is that presented in Π_{3b} (Π_{4b}).

Similarly if an edge is corrupted and then the guardian updates, new randomness will be introduced as long as the randomness is kept separate. Meaning Π_{3b} (Π_{4b}) achieves GPCS. This gives us the following advantage against G2:

$$\Pr[G2] \leq \text{Adv}^{\text{CGKA}}_{\text{CGKA}-\mathcal{B}_2}(\lambda).$$

As a result we get:

$$\text{Adv}^{\text{GCGKA}-|\text{FS,PCS}|}_{\Pi_2-\mathcal{A}}(\lambda) \leq 2\text{Adv}^{\text{ROR}}_{KDF-\mathcal{B}_1}(\lambda) + \text{Adv}^{\text{CGKA}}_{\text{CGKA}-\mathcal{B}_2}(\lambda).$$

□

A.4 Security Analysis of Π_5 and Π_6

Similarily as in Appendix A.3 we will only give the proof for Π_5 the proof for Π_6 follows exactly with only a different construction of the edge and guardian id.

Proof Theorem 1. Note that the one of the main differences between Π_5 and Π_2 is that in the latter the edge has access to the CGKA signing key. When the edge device is in limited mode it will not be able to follow group actions directly but will be reliant on the guardian forwarding messages. Only when the edge is online is there a notable difference. In that case the edge will update the group itself instead of relying on the guardian to preform the action. The proof for Theorem 1 for Π_5 is, therefore, as a result almost identical to the given in Appendix A.2.

Since Π_5 builds directly on the CGKA protocol MLS it follows directly from Definition 6 and Figure 6 that Π_5 is a GCGKA protocol, the fact that an additional randomness exist and is utilized during run time does not interfere here.

To show that Π_5 is $[FS, PCS]-$GCGKA secure we need to show that any adversary against Π_5 has negligible advantage. We perform one game hop, Figure 9, using an adversary $\mathcal{B}_1$ against CKA and then reduce the problem to an MLS adversary $\mathcal{B}_2$.

G0 - G1: As stated earlier, the game hop between G0 and G1 exchanges the correct generation of CKA-session keys with randomly selected values. Let $\mathcal{B}_1$ be and adversary that can distinguish between G0 and G1. That means $\mathcal{B}_1$ is an adversary that can determine if k is a real or random CKA session key

$$\frac{1}{2}\left|\Pr[G1]-\Pr[G2]\right| \leq \mathsf{Adv}^{\text{CKA-PCS}}_{\text{CKA}-\mathcal{B}_1}(\lambda).$$

G1 - G2: Not relevant; the games are identical.

For the final step we create an CGKA adversary $\mathcal{B}_2$ against MLS using $\mathcal{A}$ against Π_5 by running an internal version of G2 and forwarding CGKA queries to Exp^{CGKA}_{MLS}.

The validity argument presented in Appendix A.1 holds here as well.

This gives us the following advantage against G2:

$$\Pr[G2] \leq \mathsf{Adv}^{\text{CGKA}}_{\text{CGKA}-\mathcal{B}_1}(\lambda).$$

As a result we get:

$$\mathsf{Adv}^{\text{GCGKA}-|\text{FS,PCS}|}_{\Pi_2-\mathcal{A}}(\lambda) \leq 2\mathsf{Adv}^{\text{CKA-PCS}}_{\text{CKA}-\mathcal{B}_1}(\lambda)+\mathsf{Adv}^{\text{CGKA}}_{\text{CGKA-}\mathcal{B}_2}(\lambda).$$

□

enterGship$(\gamma[\mathsf{ID}],\gamma[\mathsf{ID}])$

1 : $\mathsf{k} \leftarrow\$ \mathcal{K}$
2 : **if** $\Pi = \Pi_2 \vee \Pi_5 \vee \Pi_6$
3 : $\quad \gamma^A \leftarrow$ CKA-Init-A(k)
4 : $\quad \gamma^B \leftarrow$ CKA-Init-B(k)
5 : **else**
6 : $\quad \gamma^A, \gamma^B \leftarrow \varepsilon$
7 : $(id_E, anchor, mode, getGuardian) \leftarrow (A, \mathsf{ID}, isOnline, id_G)$
8 : $(id_G, anchor, mode, getEdge) \leftarrow (B, \mathsf{ID}, isOffline, id_E)$
9 : **if** $\Pi = \Pi_2$
10 : $\quad S[id_E] \leftarrow ((\gamma^A, \mathsf{k}), \varepsilon)$
11 : **else**
12 : $\quad S[id_E] \leftarrow ((\gamma^A, \mathsf{k}), \gamma[\mathsf{ID}])$
13 : $S[id_G] \leftarrow ((\gamma^B, \mathsf{k}), \gamma[\mathsf{ID}])$
14 : **return** $id_E, (\gamma^A, \mathsf{k}), id_G, (\gamma^B, \mathsf{k})$

exitGship$(id_E, S[id_E], id_G)$

1 : $(\gamma[id_E], \gamma[\mathsf{ID}_E]) \leftarrow S[id_E]$
2 : **req** $getGuardian[id_E] = id_G$
3 : **req** $getEdge[id_G] = id_E$
4 : **req** $anchor[id_G] = anchor[id_E]$
5 : **if** $\Pi = \Pi_2$
6 : $\quad \gamma[\mathsf{ID}_E], T \leftarrow (\perp, \perp)$
7 : **else**
8 : $\quad \gamma[\mathsf{ID}_E], T \leftarrow \mathsf{rem}(\gamma[\mathsf{ID}_E], \mathsf{ID}_E)$
9 : $S[id_E] \leftarrow (\gamma[id_E], \gamma[\mathsf{ID}_E])$
10 : **return** $\gamma[\mathsf{ID}_E], T$

enterLimitedMode(id_E, id_G)

1 : **req** $(getGuardian[id_E] = id_G) \wedge (getEdge[id_G] = getEdge)$
2 : $mode[id_E] \leftarrow isOffline$
3 : $mode[id_G] \leftarrow isOnline$

exitLimitedMode (id_E, id_G)

1 : **req** $(getGuardian[id_E] = id_G) \wedge (getEdge[id_G] = getEdge)$
2 : $mode[id_E] \leftarrow isOnline$
3 : $mode[id_G] \leftarrow isOffline$

removePair$(\gamma[\mathsf{ID}_l], \mathsf{ID}, \mathsf{ID})$

1 : $\gamma[\mathsf{ID}_l], T_1 \leftarrow \mathsf{rem}(\gamma[\mathsf{ID}_l], \mathsf{ID})$
2 : **return** $\gamma[\mathsf{ID}], T_1, \varepsilon$

updateE$(id_E, S[id_E])$

1 : **req** $mode[id_E] = isOnline$
2 : $((\gamma, \mathsf{k})), \gamma[\mathsf{ID}]) \leftarrow S[id_E]$
3 : **if** $\Pi \neq \Pi_2$
4 : $\quad (\gamma[\mathsf{ID}], T_2) \leftarrow \mathsf{upd}(\gamma[\mathsf{ID}]; \mathsf{k})$
5 : **if** $\Pi = \Pi_3 \vee \Pi_4$
6 : $\quad T_1 \leftarrow KDF(\mathsf{k}, \mathtt{cntrl})$
7 : $\quad \mathsf{k} \leftarrow KDF(\mathsf{k})$
8 : $\quad \mathsf{k} \leftarrow\$ \mathcal{K}$ $\quad G2$
9 : **else if** $\Pi = \Pi_2 \vee \Pi_5 \vee \Pi_6$
10 : $\quad (\gamma, T_1, \mathsf{k}) \leftarrow\$$ CKA-S(γ)
11 : $\quad \mathsf{k} \leftarrow\$ \mathcal{K}$ $\quad G1 - G2$
12 : **if** $\Pi = \Pi_2$
13 : $\quad S[id_E] \leftarrow ((\gamma, \mathsf{k}), \varepsilon),\ T_2 \leftarrow \varepsilon$
14 : **else**
15 : $\quad S[id_E] \leftarrow ((\gamma, \mathsf{k}), \gamma[\mathsf{ID}])$
16 : **return** $S[id_E], (T_1, T_2)$

updateG$(id_G, S[id_G])$

1 : **req** $mode[id_G] = isOnline$
2 : $((\gamma, \mathsf{k}), \gamma[\mathsf{ID}]) \leftarrow S[id_G]$
3 : $(\gamma[\mathsf{ID}], T_2) \leftarrow \mathsf{upd}(\gamma[\mathsf{ID}]; \mathsf{k})$
4 : **if** $\Pi = \Pi_3 \vee \Pi_4$
5 : $\quad T_1 \leftarrow KDF(\mathsf{k}, \mathtt{cntrl})$
6 : $\quad \mathsf{k} \leftarrow KDF(\mathsf{k})$
7 : $\quad \mathsf{k} \leftarrow\$ \mathcal{K}$ $\quad G2$
8 : **else if** $\Pi = \Pi_2 \vee \Pi_5 \vee \Pi_6$
9 : $\quad (\gamma, T_1, \mathsf{k}) \leftarrow\$$ CKA-S(γ)
10 : $\quad \mathsf{k} \leftarrow\$ \mathcal{K}$ $\quad G1 - G2$
11 : $S[id_G] \leftarrow ((\gamma, \mathsf{k}), \gamma[\mathsf{ID}])$
12 : **return** $(S[id_G], (T_1, T_2)$

processUpdGE$(S[id], T)$

1 : $((\gamma, \mathsf{k}), \gamma[\mathsf{ID}]) \leftarrow S[id]$
2 : $(T_1, T_2) \leftarrow T$
3 : **if** $\Pi = \Pi_3 \vee \Pi_4$
4 : $\quad$ **req** $T_1 = KDF(\mathsf{k}, \mathtt{cntrl})$
5 : **if** $\Pi = \Pi_3 \vee \Pi_4 \vee \Pi_5 \vee \Pi_6$
6 : $\quad \gamma[\mathsf{ID}], I \leftarrow \mathsf{proc}(\gamma[\mathsf{ID}], T_2; \mathsf{k})$
7 : **if** $\Pi = \Pi_3 \vee \Pi_4$
8 : $\quad \mathsf{k} \leftarrow KDF(\mathsf{k})$
9 : $\quad S[id] \leftarrow ((\gamma, \mathsf{k}), \gamma[\mathsf{ID}])$
10 : **else if** $\Pi = (\Pi_5 \vee \Pi_6) \vee (\Pi = \Pi_2 \wedge (id = id_G))$
11 : $\quad (\gamma, \mathsf{k}) \leftarrow$ CKA-R(γ, T_1)
12 : $\quad S[id] \leftarrow ((\gamma, \mathsf{k}), \gamma[\mathsf{ID}])$
13 : **else if** $\Pi = \Pi_2 \wedge (id = id_E)$
14 : $\quad (\gamma, \mathsf{k}) \leftarrow$ CKA-R(γ, T_1)
15 : $\quad S[id] \leftarrow ((\gamma, \mathsf{k}), \varepsilon),\ I \leftarrow \varepsilon$
16 : **return** $(S[id], I)$

Figure 9: Game hops G0 and G1 used for proofs of Theorem 1 and Theorem 2 for protocols Π_2- Π_6.