The Complexities of Healing in Secure Group Messaging: Why Cross-Group Effects Matter

Version 3.0, June 2021[*]

Cas Cremers[†] and Britta Hale[‡] and Konrad Kohbrok[§]

[†]*CISPA Helmholtz Center for Information Security*
Email: cremers@cispa.de
[‡]*Naval Postgraduate School (NPS)*
Email: britta.hale@nps.edu
[§]*Aalto University*
Email: konrad.kohbrok@aalto.fi

Abstract

Modern secure messaging protocols can offer strong security guarantees such as Post-Compromise Security (PCS) [18], which enables participants to heal after compromise. The core PCS mechanism in protocols like Signal [33] is designed for pairwise communication, making it inefficient for large groups, while recently proposed designs for secure group messaging, ART [19], IETF's MLS Draft-11 [7]/TreeKEM [11], use group keys derived from tree structures to efficiently provide PCS to large groups. Until now, research on PCS designs only considered healing behaviour within a single group.

In this work we provide the first analysis of the healing behaviour when a user participates in multiple groups. Surprisingly, our analysis reveals that the currently proposed protocols based on group keys, such as ART and TreeKEM/MLS Draft-11, provide significantly weaker PCS guarantees than group protocols based on pairwise PCS channels. In fact, we show that if new users can be created dynamically, ART, TreeKEM, and MLS Draft-11 *never* fully heal authentication.

We map the design space of healing mechanisms, analyzing security and overhead of possible solutions. This leads us to a promising solution based on (i) global updates that affect all current and future groups, and (ii) *post-compromise secure signatures*. Our solution allows group messaging protocols such ART and MLS to achieve substantially stronger PCS guarantees. We provide a security definition for post-compromise secure signatures and an instantiation.

1. Introduction

Post Compromise Security (PCS) [18] is a strong security property capturing a protocol's ability to "self-heal" after a participant has been compromised. PCS is a natural counterpart to notions such as forward secrecy, but achieving it usually requires keeping state in between sessions, and therefore additional design complexity.

The Signal protocol libraries [33] achieve PCS by so-called "asymmetric ratcheting", which is a way of updating a shared symmetric key with Diffie–Hellman keying material at each message. If an adversary compromises the state of one user, they can impersonate them; meanwhile, if the user is able to generate and transfer a new DH share to their partner without adversarial interference, his subsequent symmetric keys will again be unknown to the adversary and the state is "healed".[1]

[*]: A summary of changes can be found in Section E. An extended abstract of version 3.0 appears at USENIX 2021 – this is the full version.
[‡]: The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.
[§]: This work was supported by Microsoft Research through its PhD Scholarship Programme.
1. In Signal, the asymmetric ratcheting is essentially an asynchronous variant of performing a sequence of Diffie-Hellman key exchanges, and deriving a session key from the keying material of all key exchanges processed so far. If an adversary cannot compute the DH key of even a single past DH exchange, e.g. because it only passively observed that DH exchange, it cannot derive any subsequent session keys.

Thus, PCS guarantees that if an adversary wants to eavesdrop on or impersonate a compromised participant at some future time, it must actively interfere in all of the participant's communications until that time, as well as continuing afterwards to avoid detection. This makes compromising a participant's state an unattractive attack vector.

Given the consequences and reality of credential theft [15], [21], and state actors' efforts at monitoring of mobile devices and the associated potential data loss [24], [38], the compromise scenario described above is not unrealistic. In fact, potential search of mobile devices by some state actors (which includes loss of keys) has been clearly publicized [1]; others, meanwhile, have taken a covert approach with the aim of collecting information following device access [34]. Either way, only protocols that offer PCS can automatically heal future communications following device compromise.

PCS has been analysed in the context of one-to-one communication models [17], [18], but few people in the modern era only ever message just one person (or device) at a time. With the rise of group messaging, discussion forums, and conference calls, it has become important to also consider the exact security guarantees of group protocols. This work illuminates a critical problem in group messaging, namely that *due to a lack of synchronized authenticity updates, PCS is not achieved for concurrent groups in ART and TreeKEM/MLS Draft-11, nor for future groups in any of the existing group messaging protocols, including Signal.*

Individual Groups vs. Pairwise Channels. Until recently, the protocols that offered PCS were inherently pairwise. To obtain PCS in groups, a simple solution would be to implement sending an update message m_u to a group of N members by sending it to each group member individually over $N - 1$ pairwise channels. This leads to a complete communication graph and therefore communication and computation overhead for each sender scales linearly. This also impacts any servers that buffer data until the recipients come online, requiring buffer sizes that also scale linearly in N. For large groups, which can run into thousands (e.g., for Facebook, WhatsApp, and enterprise solutions such as Cisco Webex Teams), this has effectively meant that implementations instead fall back to simpler group protocols that do not offer PCS.

The quest for efficiently achieving PCS in large groups has driven the design of new protocols. The main two designs are ART [19] and MLS Draft-11 [7], which is based on the TreeKEM protocol [11]. Notably, the ongoing development of the Messaging Layer Security (MLS) protocol is driven by the MLS working group of the Internet Engineering Task Force (IETF). ART, TreeKEM, and MLS all use tree structures to compute group keys that are continuously updated by individual group members to achieve PCS. Whereas updates for pairwise keys scale linearly, updating group keys offers logarithmic scaling, thereby improving efficiency for large groups. Thus, these group-key based protocols aim to achieve the same security guarantees, only more efficiently.

In previous academic analyses, various aspects of these designs have been explored and improvements have been suggested. However, all of these analyses only consider the security guarantees of a single group.

Security beyond one group. We perform the first analysis of the PCS healing properties of group messaging protocols that considers the interaction between multiple groups. In contrast to the previous works, our analysis reveals a significant difference in the PCS healing properties between the pairwise and group-key designs. As we will see in detail later, the underlying problem is that in the pairwise setting, the pairwise channel updates are shared across all groups, whereas updates in ART, TreeKEM, and MLS are not.

As an illustrative example, consider a party Alice from the earlier example, whose full state was compromised, e.g. during a security check at an airport. Following this, assume that while the adversary is temporarily passive, Alice sends an update message in a group G, which is received by all of its members, including Bob. Afterwards the adversary becomes active again. For both design approaches, this healing update stops the adversary from eavesdropping on G and impersonating Alice in the group G. However, there are substantial differences, including the following.
In groups based on pairwise PCS channels, we have that

- the adversary can still eavesdrop on messages received by Alice in any group G' that is not a subgroup of G, but
- the adversary can no longer impersonate Alice towards any member of G, in any group context.

In contrast, in group-key designs such as ART and TreeKEM/MLS Draft-11, we have that

- the adversary can still eavesdrop on messages received by Alice in any group $G' \neq G$, and
- the adversary can still impersonate Alice towards anyone in any group $G' \neq G$, including those not yet started, regardless of overlap in members between G and G'.

As a corollary, for the existing group-key designs, if the adversary can create new identities, it can always impersonate Alice towards Bob. Even if Alice updates all existing groups, by simply creating a new identity, e.g. Charlie, and starting a group consisting of Alice, Bob and Charlie the adversary can impersonate Alice towards Bob in that group. Thus, ART and TreeKEM/MLS Draft-11 have no mechanism to fully heal Alice w.r.t. Bob, since the adversary can always impersonate Alice to Bob in the future. This is especially surprising since the group-key designs were explicitly designed to (more efficiently) achieve the same security as the pairwise ones.

Our main contributions are the following:

1) We perform the first analysis of cross-group security effects in group messaging and their implications.
2) We identify substantial discrepancies between the security guarantees offered by state-of-the-art group messaging protocol designs that are based on pairwise channels and those that are based on group keys.
3) We map out the design space of group messaging protocols with respect to their key update mechanisms and their post-compromise security properties.
4) We propose a primitive and associated security model, post-compromise secure signatures, that enables stronger group messaging guarantees and sketch how it can be integrated in group messaging security models. To show that the definition is achievable, we provide an instantiation together with a proof of its security in our model.

Outline. In Section 2 we provide an overview of the current group messaging space, including Signal, MLS, TreeKEM, and ART, before outlining the post-compromise security differences between the main protocol types in Section 3, where we present concrete scenarios. Section 4 investigates the design space in terms of possible update options, their security effects, and relative efficiency. Based on the selected solution from Section 4, Section 5 introduces PCS secure signatures, their security, and a construction. We sketch how to construct security models for group messaging that capture cross-group security guarantees, and we consider the practical implications of the elevated security. We conclude in Section 6.

In Section A, we elaborate on the security guarantees of Signal's Sender Keys approach. In Section B, we revisit the concrete scenarios presented in Section 3, applying the solution proposed in Section 5. In Section C we provide further detail on our security model for PCS signatures, and possible future extensions. We give the full security proof in Section D.

Related Work. Since [16], [18], there has been a wave of works on post-compromise secure messaging and key exchange, including e.g., [4], [23], [26], but these works only consider pairwise communication. Some works have explored group messaging in modern protocols [35] but did not consider their PCS properties. ART [19] was the first design aiming to efficiently achieve PCS in group messaging.

Forward secure signature schemes [8] can be viewed as a related concept due to rotating the private key. In particular, forward secure signatures require that the public key remains the same over the lifetime of the key, while the private key updates. Security for such schemes is reliant on the inability of an attacker to forge signatures corresponding to any secret key sk_i, given knowledge of sk_{i^*}, where $i < i^*$. As in the general contrast between definitions of forward security and PCS, forward secure signatures are thus *backwards* focused. In comparison, PCS signatures consider the ability to heal following a compromise such that an attacker may no longer forge signatures for later signing keys. While forward secure signatures can operate by generating all certificates in advance [30], PCS signatures cannot; this serves to illustrate the divide between forward secure and PCS signatures.

Recently, there have been multiple works on continuous group key agreement (CGKA) in general and TreeKEM (or its variants) in particular. Each of the works considers CGKA in the single-group setting. [5] provided the first analysis of TreeKEM and proposed RTreeKEM, a TreeKEM version with improved FS properties. [6] studies the security of CGKAs in general and propose stronger new security notions and protocols. Another proof of TreeKEM is given in [3], which also introduces tainted TreeKEM, a modified version of the original TreeKEM with better efficiency in certain group settings.

This work explores the challenge of achieving PCS in secure group messaging, and compares possible routes towards achieving that goal. It also introduces PCS signatures and their use in terms of secure group messaging; applying ratcheting of signature keys to heal authentication has been considered in previous and concurrent works [27], [28]. Extending beyond that research, we handle PCS signatures as a separate primitive and address the context of group messaging where the signature key may be internal to the group or global as the identity. Ideas from forward secure signatures which are relevant include handling untrusted updates

[13], and modelling an unpredictable number of time periods [32]. Various other works on forward secure signatures include [2], [20], [25], [31].

2. Background

In this section, we revisit the notion of PCS and existing group messaging approaches that achieve PCS.

2.1. Threat Model

The term Post-Compromise Security (PCS) was introduced in [18] and refers to the security guarantees that can be expected *after* a (potentially complete) compromise of a party. While some designs have previously informally offered this type of guarantee, it has only been formally studied since recent works such as [17], [18].

PCS is often viewed as the counterpart to forward secrecy (FS), which is concerned with the secrecy of communication *before* the event of compromise. However, whereas FS for the most part is concerned with the protection of confidentiality of past communication, for PCS both authentication and confidentiality are immediately relevant security properties.

Generally, PCS requires an "update" operation, in which a party shares a fresh secret with its peer using a forward-secure mechanism (i.e., such that a passive adversary cannot learn the secret; using e.g., a Diffie-Hellman exchange), which they then combine with the secrets they used previously (using e.g., a Key Derivation Function). PCS is the property that, if the adversary is passive during a single update, it does not learn the update's secret, which means that the session is "healed" from that point onwards, since all later secrets depend on that secret.

Following convention, we consider adversaries that have full network access. We distinguish between two classes of adversaries: *passive* adversaries and *active* adversaries. Passive adversaries only observe the network traffic, while active adversaries can manipulate it in arbitrary ways, e.g. dropping messages, injecting their own, etc. Additionally, adversaries have the ability to compromise arbitrary parties by obtaining their full state, including long-term identity keys.

2.2. Group Messaging in Signal

The core Signal protocol is a pairwise protocol [33], in which two participants continuously send each other asymmetric ratchet updates with each message. In practice, these are ephemeral public keys of the form g^z that are then combined with previous keying material using Diffie–Hellman constructions, to finally derive symmetric keys used for message transmission. If an adversary learns the state of a party A, but the real A afterwards manages to transmit a new ephemeral public key to the peer B while the adversary is passive, then the adversary is locked out of subsequent communications. Since all future keys will depend on that update, the secret state of A is "healed".

Group messaging in the Signal libraries is done through one of two mechanisms. The first is to implement groups using the core pairwise protocol, i.e. point-to-point pairwise connections forming a complete graph over the set of group members, and the second is an alternative mechanism called *sender keys*. We summarize the main pairwise approach here (see Section A for details of using sender keys as an extension).

Using the Signal pairwise approach, a group does not correspond to a specific cryptographic mode, but is essentially a wrapper for sending each message to all group participants over individual pairwise channels. Thus, to send a message in the group $\{A, B, C, D\}$, A sends the message over the three pairwise channels $A \leftrightarrow B$, $A \leftrightarrow C$, and $A \leftrightarrow D$. Ergo, performing an asymmetric update in the group corresponds to sending three individual updates on the respective pairwise channels. If two parties share membership of multiple groups, messages in all those groups are sent over the same pairwise channel. Since pairwise channels are updated with every message sent (see [33]), we assume that when a party sends a message to a group, the pairwise channels to all other group members are updated. Authentication in Signal is based on a trust-on-first-use assumption, with no explicit ties to authentication during the protocol run [22].

2.3. Group Messaging in ART

ART was the first attempt to provide a group messaging scheme that efficiently provides PCS for large groups [19]. It uses a tree structure to represent groups, where each leaf of a tree corresponds to a group member, and in particular a party's current ephemeral public key. In general, every node in the tree is associated with a public key, where the corresponding private key is known to every member in the subtree with that node as root. This enables all members to compute the root of the tree using their private leaf key and the public keys of the nodes on the copath, which in turn allows them to compute the session key.

To start a group communication in ART, Alice generates an asymmetric "setup key" pair (s, g^s). Alice uses a member's ephemeral public key g^x (retrieved from the server) and the private setup key s to compute a Diffie–Hellman secret g^{xs} that is used as the initial asymmetric private key for that member's leaf, where $g^{\iota(g^{xs})}$ is the corresponding public key for the leaf. Doing this for each member enables Alice to compute the entire initial group tree even if the other members are currently offline. Other members can compute their initial private leaf key by combining their ephemeral private key x with the public setup key g^s. In subsequent updates, members replace the leaf values with public keys, the private key for which only they know. To achieve PCS, members update the group key by generating a new leaf key pair, and broadcasting the public key to the other members. All members then compute the new tree root key and combine it with the previous group key to obtain the new group key. Authentication is assumed in the ART design (similar to Signal), but not concretely specified.

2.4. TreeKEM/MLS Draft-11

MLS Draft-11 is an ongoing standardization effort by the IETF [7]. Similar to ART, it employs a tree-based group approach based on TreeKEM [11] for establishing and updating a symmetric group key. The tree used in MLS Draft-11 is a left-balanced binary tree. Members, leaf nodes, and public keys are associated as in ART.

To achieve PCS for a particular group, a member generates a secret string from which they derive two other secret strings – a path secret and a node secret. From the node secret, they derive a new leaf node key pair. From the path secret, they again derive a path secret and a node secret, using the node secret to derive a new key pair for their parent node, and using the path secret to continue the same process up the tree. Thus new key pairs are derived for every node on the direct path between the member's leaf node and the root. A hybrid public key encryption algorithm is used to encrypt the requisite update secrets to the nearest public key known for every member. Finally, from the root node path secret, they derive a new group secret.

Every member within an MLS group is associated with an identity public signature key and the protocol ensures that members agree on the list of group participants, as represented by the public signature keys. Messages involved in performing the update operation described above are authenticated using this signature key.

3. Multi-Group Security

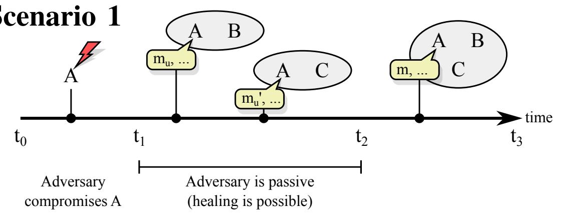
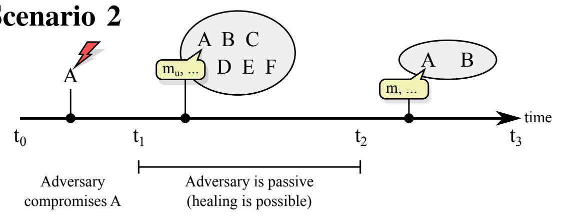
In this section we demonstrate consequences arising from the two different design approaches: Approach 1 implementing groups over pairwise channels as in Signal, and Approach 2 using group-keys as in ART/TreeKEM/MLS Draft-11. We demonstrate this using concrete scenarios. Henceforth we differentiate between keys used within groups (*local-level* keys) and those used across groups (*global-level* keys).

Recall that in the pairwise approach, if a party is a member of multiple groups, these groups do not exist independently of each another but instead rely on the same pairwise channels between their participants. In contrast, in MLS and ART, groups are independent with the possible exception of a party's signature key.

As this discussion is based on existing protocol designs, we follow the practice of Signal and MLS in assuming long-term identity keys per party and symmetric keys within groups (either pairwise or as a shared group key). Furthermore, we assume that long-term the identity keys are used to sign updates to the symmetric group keys (i.e. global-level asymmetric keys authenticate local-level symmetric key updates). For the pairwise approach, we consider PCS with regard to the aggregation of all pairwise symmetric keys between all group members. An update to a group means individual updates to all pairwise channels. Similarly, for the group-key approach, we consider PCS with regard to the actual group key, where an update operation means an update to the group key shared by the respective group. For the purposes of the following discussion, *compromise* refers to full state compromise.

For illustration purposes we consider two scenarios in both approaches. These are not exhaustive, but suffice to convey the main ideas. We introduce the two scenarios and evaluate the security of the two approaches respectively in Table 1. The following is a practical example of Scenario 2.

Table 1: Comparing the post-compromise security properties of groups built from pairwise channels or group keys

Scenario 1	**Scenario 2**
 A A B m$_u$, ... A C m$_u$', ... A B C m, ... time t$_0$ t$_1$ t$_2$ t$_3$ Adversary compromises A Adversary is passive (healing is possible)	 A A B C D E F m$_u$, ... A B m, ... time t$_0$ t$_1$ t$_2$ t$_3$ Adversary compromises A Adversary is passive (healing is possible)
A gets compromised before t_1. While the adversary is passive, A sends an update message m_u to B and an update message m'_u to C over their private channels, which is equivalent to sending in the respective groups of size two. After t_2 she sends a message m, in a group containing A, B, and C.	A gets compromised before t_1. She then sends an update message m_u in a large group (which includes B) while the adversary is passive. After t_2 A sends a regular message m just to B.
Question: Is m secure?	
Approach 1: All group communication implemented using pairwise channels with PCS	
Yes: The messages m_u and m'_u "heal" the compromise for the $\{A, B\}$ and $\{A, C\}$ channels; even when the adversary becomes active after t_2, they can no longer eavesdrop or insert data on those channels. Furthermore, since the group $\{A, B, C\}$ uses the pairwise channels, the group has now also healed. In terms of channels that are still vulnerable, the adversary can still start completely new conversations or groups after a compromise. If members do not have any previously established channels to heal, then this possibility persists.	**Yes:** Sending a message m_u to the group $\{A, \ldots, F\}$ uses, and therefore heals, all the pairwise channels between A and the other participants, and therefore also all possible groups consisting of a subset of the original group participants. Because the transmission of m uses the channel that was healed when m_u was received, m is secure. However, the healing does not include "mixed" groups comprised of a subset of the original group participants as well as other non-participants; i.e. the pairwise channels to non-participants are not healed.
Approach 2: Group channel using group keys with PCS (ART, TreeKEM, MLS Draft-11)	
No: The messages m_u and m'_u heal the pairwise channels. However, since the group with $\{A, B, C\}$ exists independently of the pairwise channels, its group key is not healed (or is completely new). Hence the adversary can learn or forge the final message m. As a corollary, if the adversary can dynamically create new users, then it can always impersonate A to B by creating a new user Z and a group consisting of A, B and Z. In the context of that group, the adversary can impersonate A towards B even after every other group had previously been udpated.	**No:** The update message m_u heals the group $\{A, \ldots, F\}$. from the compromise. However, the subgroup $\{A, B\}$ is *not* healed, and m is therefore not secure. Unlike the pairwise channel case, none of subgroups of the group updated between t_1 and t_2 is healed. Additionally, if a given subgroup did not yet exist, the adversary can create it, impersonating Alice to the newly created group.

Example 1. *Consider a large company's messaging group that includes all employees. After Alice (the CEO) is compromised, she sends a message to the entire group.*

In Approach 1, the update action implicit in the message heals all future communications with anyone in the company, including all individual channels and subgroups. However, the adversary can still impersonate Alice towards parties outside the company that Alice did not have a channel with at the time of compromise. In Approach 2 (ART, MLS Draft-11), the update action heals only the group and the adversary can continue to impersonate Alice to other employees on pairwise channels until Alice has sent an update message to each of them individually. Additionally, even a passive adversary can continue to eavesdrop on all other groups that Alice was already in (including pairwise channels) until they are updated individually. Finally and most importantly, even after sending updates to every existing subgroup and every individual channel, the adversary can still create new subgroups and impersonate Alice towards anyone in those subgroups.

If we only consider a *passive adversary*, we can already show a significant difference between the approaches. Suppose Alice is in groups $G^1, \ldots, G^N$ when she is compromised. In Approach 1, once Alice sends a single update message to the supergroup $G = \bigcup_{i=1}^{N} G^i$ (or to several groups as long as their union contains G), she is completely healed with respect to the passive adversary. The underlying reason is that such updates cause an update of all the underlying pairwise channels, thereby essentially replacing all encryption keys that the attacker might have learned during the compromise. In Approach 2, Alice is only completely healed once she has sent update messages in *all* groups $G^1, \ldots, G^N$. Until that point, the adversary can still eavesdrop on all groups in which she has not yet sent an update message.

For an *active adversary*, the situation is worse. In Approach 1, sending an update to G also heals all future groups that are subgroups of G, even if they were not yet active during the compromise. In Approach 2, while the group key of G can be updated, the re-use of authentication keys implies that the adversary remains capable of impersonating Alice in any group unless there has been an explicit update to that group. This notably includes new groups created by the adversary, where they can include Alice and any other party it wishes to impersonate Alice towards.

These observations point to the fact that in Approach 2, PCS per the original definition [18] is not achieved in these approaches due to the *scope of the update*: in Approach 1, each update message heals a previously compromised sender with respect to the recipient for all future communications between them, irrespective of the context. In contrast, in the group proposals in ART and MLS Draft-11, a previously compromised sender's update message only heals the group context it was sent to.

4. Design Space

The previous analysis raises the obvious question: is it possible to provide efficient group messaging solutions that offer PCS with similar (or even better) healing properties than the pairwise solution? Our aim is to extend the group-key based approach from ART and TreeKEM with additional PCS mechanisms to achieve stronger guarantees at minimal cost.

The underlying problem of the scenarios in the previous section is one of scope: the updates to the group keys only healed the specific group, whereas the updates to pairwise keys healed those channels in all groups.

4.1. Design space elements

In this section we explore the design space of updating keys. We consider two main axes: which keys are updated, and how their updates are scheduled.

In the current designs for ART and TreeKEM, only group keys are updated, and in Signal's pairwise channels for groups, only pairwise channel keys are updated. There is no *a priori* reason to only update a single key type, and we may consider updates to symmetric group keys and asymmetric data signing keys separately. Furthermore, one may consider additional (identity) signature keys used to authenticate parties – signature keys that are not bound to a particular group. In MLS, for example, updatable group-specific signature keys have been considered [10]. Consequently, we choose to explore designs that may include any of these three key types, and which may update them individually or jointly.

We will differentiate between two main scheduling options for key updates. The first is scheduling on the basis of communication activity, e.g., after N messages were sent. We can further subdivide this option: we can schedule on the basis of communications in a single group or in all groups. The second is scheduling independently of communication activity, e.g., after T time units.

4.2. Informal analysis

We summarize our informal analysis in Table 2a and Table 2b and explain the considerations in more detail below.

Table 2a compares updates for healing of confidentiality. In the first column we have updates to a group-specific symmetric encryption key, gk^i, triggered by activity within the same group. Since the effect is internal to the group, PCS is broken in concurrent groups (2nd Group Attack), the healing does not affect future groups (NF), and the computational overhead is $\mathcal{O}(1)$ in the minimum number of keys that must be updated by a party A (see Table 2a).

In the second column, we see the case that update of gk^i is triggered by activity (such as the number of messages A has sent) within any group, causing leakage of activity frequency information in other groups (InfoLeak). For example, if A messages frequently in group i, then all groups will be frequently updated; however, if A has little activity in group j, then other members of group j can deduce information about A's activity level in other groups, breaking privacy. While this implies that all groups of a party are updated simultaneously, preventing concurrent group attacks, healing does not affect future groups. The computational overhead mirrors the number of groups that must be updated.

Using a time-based update trigger, the third column considers updates by epochs. As these updates apply to all current groups (but not future groups, hence NF), the computational overhead is linear in the total number of groups the party is a member in. As the update is unrelated to group activity, it does not leak information.

Critically, all column options do not apply PCS healing to future groups. This is due to the fact updates are *local* to groups and therefore only apply to pre-existing ones. Likewise, since updates are to symmetric keys only, authenticity is not healed in any column.

Table 2b compares updates for healing of authenticity, where authentication keys may be asymmetric and group-specific, or global identity keys. Updates to a global signature key achieve PCS for authenticity with a single update (row two) vice the $\mathcal{O}(N)$ computational overhead of per-group updates. Notably, global updates apply PCS to future groups as well.

4.3. Global-level vs. Local-level Updates

The discussion above points towards a conclusion that differentiates between local-level and global-level updates. Forward Secrecy (FS) for the group key, as described in Section 2, considers attacks at the local-level (compromise of session key) while PCS per the original definition [18] is about attacks at both the local-level and global-level (compromise of the session and identity keys). Thus, to achieve local-level guarantees (e.g. group key FS), it is logically only necessary to update keys at that level, while achieving global-level guarantees (e.g. PCS) it is necessary to update global-level keys (i.e. identity keys). Global-level key updates include aggregated updates to all group keys owned by a party (i.e. epoch updates of either symmetric or asymmetric keys) or direct updates to the asymmetric identity keys. Reduction of overall computational overhead subsequently underscores the relative benefit of updating only the asymmetric identity keys.

It is possible to achieve single group-specific PCS without updating global-level keys, as in MLS Draft-11. However, the PCS guarantees then only apply to the specific group and do not extend to other groups a party A may be a member in, leaving collateral and residual effects (see Section 3).

4.4. Sharing randomness across updates

To improve efficiency, one could argue for re-use of updates among groups or among scopes (local/global), in order to prevent concurrent group attacks while simultaneously avoiding an $\mathcal{O}(N)$ overhead. However, this entails reusing keying material – e.g. a party uses the same keying material from an update in one group to update their other groups (or to update global keys such, as identity signature keys). We discard this optimization on two grounds. First, the updates sent in ART and TreeKEM need to be encrypted individually for the nodes on the copath in a specific group – which means that no bandwidth is saved by re-using update material, and updates are linear in the number of groups. Second, from a secure design perspective, and to simplify proofs, we want to maintain maximum key separation.

	Scheduling Updates for Confidentiality Keys		
	Communication Activity Related		**Periodic**
Keys updated	Within group i	Within any group	Epoch-based
Sym. group key: gk^i	Heal conf: group i 2nd Group Attack/NF $\mathcal{O}(1)$ Sym. Updates	Heal conf: all groups InfoLeak/NF $\mathcal{O}(N)$ Sym. Updates	Heal conf: all groups NF $\mathcal{O}(N)$ Sym. Updates

(a) Design space: updated keys, scheduling updates, and consequences for confidentiality healing. (Legend in Table 2c)

	Scheduling Updates for Authentication Keys		
	Communication Activity Related		**Periodic**
Keys updated	Within group i	Within any group	Epoch-based
Asym. group key pair: (sk^i_A, pk^i_A)	Heal auth: group i 2nd Group Attack/NF $\mathcal{O}(1)$ Asym. Updates	Heal auth: all groups InfoLeak/NF $\mathcal{O}(N)$ Asym. Updates	Heal auth: all groups NF $\mathcal{O}(N)$ Asym. Updates
Asym. global key pair: (sk_A, pk_A)	▨	Heal auth: all groups + F InfoLeak $\mathcal{O}(1)$ Asym. Updates	Heal auth: all groups + F $\mathcal{O}(1)$ Asym. Updates

(b) Design space: updated keys, scheduling updates, and consequences for authentication healing. (Legend in Table 2c)

Notation	**Explanation**
gk^i	Symmetric key for a group i.
(sk^i_A, pk^i_A)	A's asymmetric key pair for a group i.
(sk_A, pk_A)	A's asymmetric global key pair (identity key).
▨	Non-functionality, e.g. when a global update impacts only a particular group.
InfoLeak	Updates which are not set to occur at an identity-level fixed epoch regularity leak information about individual group-level activity. If updates are set to occur at an identity-level fixed epoch regularity, then PCS is achievable at the specified epoch frequency.
2nd Group Attack	PCS is broken under a second group (e.g. concurrent group or sequential group). A's update in group G^i does not apply to other groups $\{G^1, \ldots, G^N\} \setminus \{G^i\}$.
NF	No future groups (which A has yet to join) achieve PCS following the specified key updates.
$+$ F	Future groups (which A has yet to join) achieve PCS following the specified key updates.
$\mathcal{O}(N)$ sym. (resp. asym.) key updates	An update with the given *Additional effects* requires $\mathcal{O}(N)$ update calculations of symmetric (resp. asymmetric) keys.
$\mathcal{O}(1)$ sym. (resp. asym.) key updates	An update with the given *Additional effects* requires $\mathcal{O}(1)$ update calculations of symmetric (resp. asymmetric) keys. If a calculation requires 1 symmetric and 1 asymmetric key update, we consider the higher computational cost, i.e. $\mathcal{O}(1)$ asymmetric key updates.

(c) Legend for Section 4

Table 2: Design space: summary of informal analysis

4.5. Optimizing Update Type Selection for Confidentiality and Authenticity

When authenticity PCS is achieved via periodic asymmetric key updates, (i.e. optimizing selection in Table 2b, we achieve the added benefit of future group PCS and protection against concurrent group attacks. It is therefore possible to combine this option with that of either the first or third column of Table 2a, to PCS heal both authenticity and confidentiality – covering future groups and without information leakage or concurrent group attacks.

If authentication has been healed, an adversary may not inject updates into any group, but may continue to eavesdrop until confidentiality has also been healed. Thus, an epoch-based confidentiality healing solution may be ideal (column three), but it is also an option to optimize on computational overhead and update groups ad-hoc dependent on activity levels (column one). If a party is a member of many, albeit largely inactive groups, this option may be of particular interest. The computational overhead of these solution combinations would be $\mathcal{O}(1)$ asymmetric updates, and either $\mathcal{O}(N)$ epoch-based symmetric updates or $\mathcal{O}(1)$ ad-hoc symmetric updates.

4.6. Update Frequency by Type

Assessing the appropriate relative frequency of update type (i.e. of asymmetric signature keys or symmetric group keys) depends on the overall security goals. Here we consider the consequences of different update types (for confidentiality/authenticity) following a compromise, before aggregating that view over all possible compromise epochs. We assume the scenario of the following update combination for achieving authenticity and confidentiality in a group, following from the discussion in Section 4.2 and optimizing for efficiency: a global signature key pair (sk, pk) with epoch-based updates and a symmetric group key gk^i with group i internal activity-related updates.

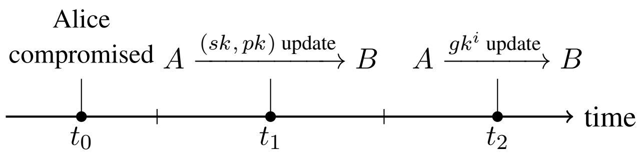

Alice compromised
$A \xrightarrow{(sk,pk)\text{ update}} B$
$A \xrightarrow{gk^i\text{ update}} B$
time
t_0
t_1
t_2

Figure 1: Update order: signature key first.

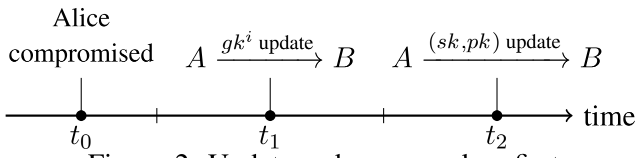

Alice compromised
$A \xrightarrow{gk^i\text{ update}} B$
$A \xrightarrow{(sk,pk)\text{ update}} B$
time
t_0
t_1
t_2

Figure 2: Update order: group key first.

Case 1: Update (sk, pk) before gk^i. Suppose that an adversary $\mathcal{A}$ compromises Alice at time t_0 per Fig. 1, followed sequentially by an update to sk at some later point, and finally an update to gk^i.

$t_0 - t_1$: The entire state of Alice is compromised.
$t_1 - t_2$: $\mathcal{A}$ can act as a passive attacker in any group G^j which Alice was a member of at t_0, but cannot act as an active attacker.
$t_2 \rightarrow$: Alice is healed in group i. $\mathcal{A}$ can still act as a passive attacker in groups $\{1 \ldots, N\} \setminus \{i\}$ until gk^j is healed.

Case 2: Update gk before sk. Suppose that Alice is compromised at time t_0 per Fig. 2 as in Case 1, but instead updates the gk^i before (sk, pk).

$t_0 - t_1$: The entire state of Alice is compromised.
$t_1 - t_2$: $\mathcal{A}$ cannot act as a passive attacker in group G^i, but can act as a passive attacker in any group $\{1, \ldots, N\} \setminus \{i\}$ and as an active attacker in group i.
$t_2 \rightarrow$: If $\mathcal{A}$ has not injected updates between t_1 and t_2, Alice is healed in group i. $\mathcal{A}$ can still act as a passive attacker in groups $\{1, \ldots, N\} \setminus \{i\}$ until gk_j is healed.

In consequence of the above cases, updates to group encryption keys provide a short-term confidentiality-focused solution, blocking out passive attackers in a specific group. They do not provide a long-term confidentiality-focused solution, since an attacker can still impersonate Alice, and provide updates in other existing and future groups. Updates to signature keys, on the other hand, provide a long-term authenticity-focused solution, ensuring that any encryption key updates following the signature key update are valid. This validity then propagates to benefit long-term confidentiality through the use of updated encryption keys.

Some applications may prioritize immediate message privacy in a particular group i over long-term effects in all groups $\{1, \ldots, N\}$, or may enforce epoch-based group updates for $\{gk^1, \ldots, gk^N\}$. Nonetheless, "locking out" an attacker as in Case 1 is arguably preferred, given the cost/benefit balance of confidentiality in a single group vice blocking impersonation.

In practice, it is not usually possible to know the point of compromise t_0. Consequently, let $n_{(sk,pk),t}$ be the number of updates made to (sk, pk) in a given time period t and let $n_{gk^i,t}$ be the number of updates made to gk^i in the same time period. If $n_{(sk,pk),t} > n_{gk^i,t}$ it follows that, for a random compromise of Alice in t, the probability of achieving Case 1 security is higher than achieving Case 2 security. If $n_{gk^i,t} > n_{(sk,pk),t}$, the converse holds.

5. Solution and High-Level Comparison

Our analysis points to two important aspects of a solution:

1) global-level updates are necessary for effective PCS group healing comparable to pairwise messaging solutions, and
2) secure updates to identity keys are necessary to ensure authentication after compromise, ideally across groups.

In this extended abstract, we focus on the second point, and address the first point in detail in the extended version. At the end of this section we will nevertheless sketch how these aspects together may achieve stronger PCS guarantees.

Concretely, we first introduce the notion of a *Ratcheting Digital Signature* (RSIG) scheme, which formally captures the type of rotating signatures we require, and the associated *Post-Compromise Secure Signature* (PCS-SIG) security notion. We then provide an RSIG construction based on an EUF-CMA secure signature scheme, and provide a security proof in Appendix D. Finally, we show how this solution can be used to achieve the PCS guarantees aimed for in the goals of MLS and ART.

5.1. Post-Compromise Secure Signatures

Forward-secure signatures were first proposed by Bellare and Miner [9]. However, as with the general distinction between FS and PCS, the security demands on the signature scheme proposed above show a different security goal than FS. Consequently we propose PCS-Signatures. We instantiate it using an existentially unforgable signature scheme that effectively ratchets its keys. It differs from the key evolving signatures used for forward-secure signatures in that not only the secret but also the public key is updated. Subsequently we introduce the PCS-Signature security game.

Definition 1 (Ratcheting Digital Signatures)**.** *A* ratcheting digital signature *scheme is a tuple of algorithms* $\mathrm{RSIG} = (\mathrm{RSIG.Gen}, \mathrm{RSIG.Update}, \mathrm{RSIG.RcvUpdate}, \mathrm{RSIG.Sign}, \mathrm{RSIG.Verify})$*, where:*

- $\mathrm{RSIG.Gen}$ *is a probabilistic key generation algorithm which takes as input a security parameter* $\lambda \in \mathbb{N}$ *and returns a pair* (sk, pk)*, where* sk *is the initial secret key and* pk *is the initial public key.*
- $\mathrm{RSIG.Update}$ *is a (possibly probabilistic) key update algorithm which takes as input a secret and public key pair* (sk, pk) *and returns a new secret and public key pair* (sk', pk') *and an update message* m_u*.*
- $\mathrm{RSIG.RcvUpdate}$ *is a deterministic update receiving algorithm which takes as input a public key* pk *and an update message* m_u*, and returns the updated public key* pk'*.*
- $\mathrm{RSIG.Sign}$ *is a (possibly probabilistic) signing algorithm which takes as input the secret key* sk *and a message* m *and returns* σ*, a signature on* m*.*

- RSIG.Verify *is a deterministic verification algorithm which takes as input a public key pk, a message m, and a signature σ_m and outputs bit* verify *such that* $\text{verify} = 1$ *if σ_m is a valid signature of* m *and* $\text{verify} = 0$ *otherwise.*

  Correctness. We require that

- For any sequence of i calls to RSIG.Update and RSIG.RcvUpdate, if
  - $\big((sk_i, pk_i), m_u\big) \xleftarrow{\$} \text{RSIG.Update}(sk_{i-1}, pk_{i-1})$, and
  - $pk' \leftarrow \text{RSIG.RcvUpdate}(pk_{i-1}, m_u)$,

  then $pk_i = pk'$.
- $\text{RSIG.Verify}(pk, m, \text{RSIG.Sign}(sk, m)) = 1$ for every message m and any pair (sk, pk) with probability 1.

```
PCS-SIG.SignerGen()
assert $sk_1 = \perp$
sqn ← 1
$sk_{\mathsf{sqn}}, pk_{\mathsf{sqn}} \leftarrow\$ \text{RSIG.Gen}$
return $pk_{\mathsf{sqn}}$
```

```
PCS-SIG.VerifierGen()
assert $pk_1 \neq \perp$
assert verifier.pk $= \perp$
verifier.pk $\leftarrow pk_1$
return verifier.pk
```

```
PCS-SIG.Corrupt($i$)
assert $sk_i \neq \perp$
Corrupted ← Corrupted $\cup \{i\}$
return $sk_i$
```

```
PCS-SIG.Update()
assert $sk_{\mathsf{sqn}} \neq \perp$
$((sk_{\mathsf{sqn}+1}, pk_{\mathsf{sqn}+1}), m_u) \leftarrow$
    $\text{RSIG.Update}(sk_{\mathsf{sqn}}, pk_{\mathsf{sqn}})$
sqn ← sqn + 1
UpdateList ← UpdateList $\cup \{(\mathsf{sqn}, m_u)\}$
return $(pk_{\mathsf{sqn}}, m_u)$
```

```
PCS-SIG.RcvUpdate($m_u$)
assert verifier.pk $\neq \perp$
$pk' \leftarrow$ RSIG.RcvUpdate(verifier.pk, $m_u$)
if $pk' \neq \perp$ then
    RcvdUpdateList ←
        RcvdUpdateList $\cup \{m_u\}$
    verifier.pk $\leftarrow pk'$
return verifier.pk
```

```
PCS-SIG.Sign($m$)
assert $sk_{\mathsf{sqn}} \neq \perp$
$\sigma \leftarrow \text{RSIG.Sign}(sk_{\mathsf{sqn}}, m)$
MsgList ← MsgList $\cup \{(m, pk_{\mathsf{sqn}})\}$
return $\sigma$
```

```
PCS-SIG.Verify($m, \sigma$)
assert verifier.pk $\neq \perp$
verify ← RSIG.Verify(verifier.pk, $m, \sigma$)
if verify $= true$ $\wedge$ ($m$, verifier.pk) $\notin$ MsgList
    $\wedge$ ($\forall i \in$ Corrupted, $\exists (j, m_u) \in$ UpdateList s.t. $j > i$
        $\wedge$ $m_u \in$ RcvdUpdateList) then
    return $b$
return verify
```

Figure 3: Description of the oracles provided by $\mathsf{PCS\text{-}SIG}^{b,\mu}$, where $b \in \{0, 1\}$ and μ is an RSIG scheme.

Definition 2 (PCS-SIG Security). *Let μ be an RSIG scheme. Then we say μ is PCS-SIG secure with adversarial advantage function $\epsilon_{\mathsf{PCS\text{-}SIG}}(\cdot)$, if for all adversaries $\mathcal{A}$ we have that*

$$\mathsf{PCS\text{-}SIG}^{0,\mu} \overset{\epsilon_{\mathsf{PCS\text{-}SIG}}(\mathcal{A})}{\approx} \mathsf{PCS\text{-}SIG}^{1,\mu},$$

where for $b \in \{0, 1\}$ the oracles provided by $\mathsf{PCS\text{-}SIG}^{b,\mu}$ *are defined in Figure 3.*

The above notion of PCS-Signature security allows the adversary to initialize, and interact with a signer and a verifier. After using SignerGen to initialize the signer with their initial keys, the adversary can have the signer issue signatures for arbitrary messages using Sign and update the signer's current signature key using Update. The adversary can also obtain the signer's secret key by issuing a Corrupt query. Similarly, after using VerifierGen, which initializes the verifier's public key verifier.pk with the initial signer key generated by the SignerGen oracle, the adversary can have the verifier verify signatures using the Verify oracle and update the verifier's verifier.pk using the RcvUpdate oracle. The adversary wins the game, if they call the Verify oracle with input (m, σ) and the following conditions are met:

- the signature is successfully verified using the verifier's current public key,
- σ was not the output of a query to Sign with input m,
- each compromise has been healed: after every compromise, the adversary has at least once used an update message output by the Update oracle to successfully update the verifier's public key using the RcvUpdate oracle.

Thus, to win the game, the adversary is not required to submit general forgeries, but a forged signature *following* at least one successful update between signer and verifier after each compromise. We use an indistinguishability notion in the model to ease composition with other security models. In this notion the adversary distinguishes between two games by creating a forgery that causes the game to return the bit b. For further discussion, see Appendix C.

5.2. RSIG from an EUF-CMA Secure Signature Scheme

We now introduce an RSIG construction from an EUF-CMA-secure signature scheme. Intuitively, the RSIG key pair can be simply the signature key pair, while signing and verifying work as in a standard signature scheme. To create an update message from a given signing key, the construction generates a new key pair and signs the public key, returning the public key and the signature as the message and the new signing key as the corresponding RSIG private key.

To prevent the adversary from forging update messages, regular signatures and update messages must be easily distinguishable. We achieve this by appending the string "update" to public keys when signing them to create update messages and similarly appending "msg" when signing regular messages. These can be handled via a uniquely decodable bit position added to the signed data.

```
$\mu$.Gen($\lambda$)
$(pk, sk) \leftarrow \nu.\mathrm{Gen}(\lambda)$
return $(pk, sk)$

$\mu$.Sign($sk, m$)
$\sigma \leftarrow \nu.\mathrm{Sign}(sk, (m||\text{“msg”}))$
return $\sigma$

$\mu$.Verify($pk, m, \sigma$)
verify $\leftarrow \nu.\mathrm{Verify}(pk, (m||\text{“msg”}), \sigma)$
return verify

$\mu$.Update($sk, pk$)
$(pk', sk') \leftarrow \nu.\mathrm{Gen}(\lambda)$
$\sigma \leftarrow \nu.\mathrm{Sign}(sk, (pk'||\text{“update”}))$
$m_u \leftarrow (pk', \sigma)$
return $((sk', pk'), m_u)$

$\mu$.RecvUpdate($pk, m_u$)
$(pk', \sigma) \leftarrow m_u$
verify $\leftarrow \nu.\mathrm{Verify}(pk, (pk'||\text{“update”}), \sigma)$
if verify $= true$ then
    return $pk'$
else
    return $\perp$
```

Figure 4: Description of RSIG construction.

We provide a more detailed explanation and discussion of the PCS-SIG security model as well as considerations regarding the RSIG construction in Section C.

Construction 1 (RSIG Construction)**.** *Let $\nu = (\mathrm{Gen}, \mathrm{Sign}, \mathrm{Verify})$ be an EUF-CMA-secure signature scheme with security parameter λ. Then we can construct an RSIG scheme μ as described in Figure 4.*

Proof overview. The goal of our proof is to reduce the PCS-SIG-security of the RSIG construction μ to the EUF-CMA-security of the digital signature scheme ν. We begin by "lifting" EUF-CMA twice: First, we introduce corruptible EUF-CMA (CEUF-CMA), which adds a new oracle to EUF-CMA that allows the adversary to corrupt the signer and thus retrieve the secret key. However, after corrupting the signer, the verification oracle is disabled such that the adversary can not win anymore by submitting valid forgeries. We then lift CEUF-CMA to its multi-instance version MI-CEUF-CMA, which allows the adversary to interact with $n \in \mathbb{N}$ instances of CEUF-CMA and which was first introduced by Abdalla, Benhamouda and Pointcheval in [2] (although for SUF-CMA instead of EUF-CMA). For both lifts, we provide straight-forward reductions to EUF-CMA, yielding a combined security loss of factor n. Finally, we build a reduction $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ to reduce the PCS-SIG security of our construction to the MI-CEUF-CMA security of the underlying signature scheme.

Using that reduction, we prove that for all adversaries $\mathcal{A}$:

$$\epsilon_{\mathsf{PCS\text{-}SIG}}(\mathcal{A}) = \epsilon_{\mathsf{MI\text{-}CEUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R}_{\mathsf{PCS\text{-}SIG}}),$$

We give the full security proof in Section D.

5.3. Group Messaging Security Models

We now outline how, given a base security model for group messaging with an uncompromisable authentication layer, we can construct security models that capture the various cross-group security guarantees based on our observations.

Security models for modern secure messaging protocols are extremely intricate even for two-party protocols and under the assumption of an uncompromised authentication layer [17]. This complexity is due to the variety and quantity of key combinations involved in such protocols, and dynamic aspects inherent in epoch-based changes. A detailed security model for a group protocol such as MLS is the subject of ongoing investigations, but will inevitably be even more complex.

As most analyses of group messaging protocols use a game-based security model with uncompromised authentication layer, we formalize the orthogonal extensions necessary to capture the various cross-group security guarantees we observed. We assume the base model has a so-called freshness predicate that we call $\mathit{Fresh}_{\mathit{base}}$. In game-based security models, a freshness predicate[2] is commonly used to identify which local sessions (and in particular, their session keys) can be expected to be secure. Note, that a session corresponds to a party's local group state, including the group key.

Capturing our scenarios requires two main changes to the base model:

1) Adding an adversary query that models the compromise of *all* secrets, including the secrets for the authentication layer (such as identity keys).
2) Adapting the $\mathit{Fresh}_{\mathit{base}}$ predicate to model the specific security guarantees expected.

For the first item, we add a query $\mathit{FullCompromise}(X)$ that returns the full state of a party X, including the secrets used for authentication, such as signature keys in MLS. For the second change we construct several new freshness predicates. We define the following notation for a session:

- sid: a variable identifying a particular (local) session.
- $\mathit{sid.actor}$: a variable identifying the party that session sid belongs to.
- $\mathit{sid.G}$: the set of parties in this session's group.
- $\mathit{Fresh}_{\mathit{base}}(\mathit{sid})$: the freshness condition of the given base security model for a session sid.

When evaluating a freshness predicate, we say that a party X *was revealed*, if and only if there was a reveal query on X or on one of its sessions; this notably includes $\mathit{FullCompromise}$ and any reveal queries from the base model.

The main observation that underlies post-compromise security guarantees is that in theory, a party can *heal* from a compromise if it afterwards generates new randomness, successfully transmits this to its communication partners, and it is integrated into future keying material. All protocols that provide PCS have such *update* mechanisms. To specify security guarantees, we must formalize whether a healing update has happened: that is, whether a post-compromise update by an honest party was received by another session, which notably requires the adversary to be passive and not modify the message in transit – this temporary passivity enables healing and is a component of any PCS model. To capture this, we define the $\mathit{GrpUpd}_{\mathit{passive}}$ predicate, which indicates that an update to the group symmetric key(s) was processed.

Definition 3 ($\mathit{GrpUpd}_{\mathit{passive}}$)**.** *For a session sid and a party $A \in \mathit{sid.G}$, the predicate $\mathit{GrpUpd}_{\mathit{passive}}(A, \mathit{sid})$ holds iff:*

- *A was never revealed, or*
- *A was last* revealed *at time t and generated an update (or join) to* symmetric group *key(s) with randomness r at time t' ($t' > t$), and this update with r was processed by the local session sid.*

We can now propose Freshness predicate that captures the expected cross-group security properties of MLS Draft-11:

Definition 4 ($\mathit{Fresh}_{\mathit{MLSDraft-11}}$)**.** *For session identifier sid, we define $\mathit{Fresh}_{\mathit{MLSDraft-11}}(\mathit{sid})$ as:*

$$\big(\mathit{Fresh}_{\mathit{base}}(\mathit{sid}) \land \neg\exists X \;:\; \mathit{FullCompromise}(X)\big) \lor$$
$$\big(\forall X \in \mathit{sid.G} \;:\; \mathit{GrpUpd}_{\mathit{passive}}(X, \mathit{sid})\big) \;.$$

2. For example, for models in which the adversary has a session-key-reveal query, sessions whose session key was revealed are not considered fresh, because we cannot expect them to be secure; yet we expect some form of security for sessions whose keys were not revealed, and hence they are considered fresh. This is is sometimes expressed as "sessions are fresh if there is no trivial attack against them".

This freshness predicate has two parts: the first line captures that a session is fresh if it was so in the base model (where $\mathit{FullCompromise}(X)$ is not yet defined). Additionally, a session is now also fresh if it has processed symmetric key group updates from each group member that was previously revealed, during which the adversary was passive.

Recall that the Freshness predicate identifies sessions for which we obtain a security guarantee. Thus, in Definition 4, if a full compromise of X occurs, then sessions for which X is a group member can only be healed by updates from X that are specific to the session. To regain security in all sessions after such a full compromise, the adversary must be passive for updates or joins from X in all (both *current* and *future*) sessions of which X is (or will be) a member.

In contrast, groups based on pairwise channels can offer stronger guarantees, because they effectively share updates among groups. For groups based on pairwise channels such as Signal, we expect the following Freshness condition:

Definition 5 ($\mathit{Fresh}_{\mathit{Signal}}$)**.** *For session identifier sid, we define $\mathit{Fresh}_{\mathit{Signal}}(\mathit{sid})$ as:*

$$\begin{aligned}
&\big(\mathit{Fresh}_{\mathit{base}}(\mathit{sid}) \wedge \neg\exists X \ : \ \mathit{FullCompromise}(X)\big) \vee \\
&\big(\forall X \in \mathit{sid}.G \ : \\
&\ \exists \mathit{sid}' \ : \ \mathit{sid}'.\mathit{actor} = \mathit{sid}.\mathit{actor} \wedge \mathit{GrpUpd}_{\mathit{passive}}(X, \mathit{sid}')\big) \ .
\end{aligned}$$

The first line is identical to before. The remainder encodes that a session can also be healed if a party (say, A) receives honest updates from each revealed participant X of the group, in the context of *any* session sid' of A. This leads to a strictly stronger guarantee: any session considered fresh for MLS Draft-11 is also fresh for this definition, where $\mathit{sid} = \mathit{sid}'$.

The difference between the two preceding freshness conditions in practice is that a group based on pairwise channels can be healed by updates in *other* groups (e.g. in a subgroup, or even in a supergroup by a suitable union of groups), as we saw in the scenarios in Section 3.

The attack scenarios in Table 1 show severe drawbacks to only requiring Definition 4 or 5 as the freshness condition for the concurrent group setting under real-world attacks. We next show how to construct a stronger freshness predicate and sketch how PCS-SIG can achieve it.

5.4. Applying PCS-SIG to Group Messaging Security Models

We now analyse how we can use a PCS-SIG secure signature scheme to improve the security of group messaging protocols. In particular, in absence of an existing model for secure group messaging that considers multiple groups, we will make an argument how Definitions 4 and 5 would change if PCS-SIG were integrated into their corresponding model.

In terms of freshness conditions, we argue that a successful PCS-SIG Update operation can be expressed as follows:

Definition 6 ($\mathit{AuthUpd}_{\mathit{passive}}$)**.** *For two parties A, B, the predicate $\mathit{AuthUpd}_{\mathit{passive}}(A, B)$ holds iff:*

- *A was never revealed, or*
- *A was last* revealed *at time t and generated an update to* authentication *key(s) with randomness r at time t' ($t' > t$), and this update with r was processed by B.*

Notably, the $\mathit{AuthUpd}_{\mathit{passive}}$ predicate corresponds exactly to the condition in $\mathsf{PCS\text{-}SIG}^{b,\mu}$ under which the adversary has to submit a forgery to win the game, thus describing the guarantees that PCS-SIG adds to the base model.

As discussed in Section 4.6, appropriately updating signature keys allows healing of the authentication layer, which means that a GrpUpd action no longer requires the adversary to be passive (at least until the next compromise), which we express as follows:

Definition 7 ($\mathit{GrpUpd}_{\mathit{active}}$)**.** *For a session sid and a party A, the predicate $\mathit{GrpUpd}_{\mathit{active}}(A, \mathit{sid})$ holds iff:*

- *A was never revealed, or*
- *A was last* revealed *at time t, and session sid processed an update to* symmetric group *key(s), assuming that the update was generated at time t' ($t' > t$) by A.*

In contrast to the passive variant, this predicate holds even if an active adversary created or modified the received update.

MLS Freshness. In Definition 4, where for a group key to be considered fresh (i.e. expected to be secure), either it would have to be considered fresh in the base model without any compromise of the authentication key, or after the most recent compromise of a member, that member would have to have performed a successful update ($\mathit{GrpUpd}_{\mathit{passive}}$). For a composition of the base model with a PCS-SIG secure signature scheme, we extend the definition as follows.

Definition 8 ($\mathit{Fresh}_{\mathit{MLS}+\mathit{PCS\text{-}SIG}}$)**.** *For session identifier sid, we define* $\mathit{Fresh}_{+\mathit{PCS\text{-}SIG}}(\mathit{sid})$ *as:*

$$\begin{aligned}&\big(\mathit{Fresh}_{\mathit{base}}(\mathit{sid}) \wedge \neg\exists X \ : \ \mathit{FullCompromise}(X)\big) \vee \\ &\big(\forall X \in \mathit{sid}.G \ : \ \mathit{GrpUpd}_{\mathit{passive}}(X, \mathit{sid})\big) \vee \\ &\big(\forall X \in \mathit{sid}.G \ : \\ &\ \ \mathit{AuthUpd}_{\mathit{passive}}(X, \mathit{sid}.\mathit{actor}) < \mathit{GrpUpd}_{\mathit{active}}(X, \mathit{sid})\big)\end{aligned}$$

where we abuse notation and write $E < E'$ to denote that either X was never revealed, or otherwise the update for E occurred before the update at E'.

This definition is a straightforward extension of Definition 4, adding a third guarantee: a session is also fresh if, for each revealed member of the group, the adversary is passive during a single update to authentication key(s), which is then followed by an update to the symmetric group key(s) by X. Notably, for the update to symmetric group key(s), the adversary is *not* required to be passive to achieve freshness. This condition handles the specific case of an PCS-SIG Update preceding the symmetric group update – Section 4.5 and 4.6 compare this with the alternative ordering.

This enables the guarantees from MLS Draft-11 freshness, but also provides healing if the attacker is passive during a single update to authentication key(s). As such we are able to protect against the attacks described in Section 3 as well as providing security in additional scenarios (see Section B for examples). While the healing in the sense of confidentiality only becomes effective after the group update (or join), we do not require the adversary to be passive for the symmetric group-specific actions. Note, that the $\mathit{AuthUpd}_{\mathit{passive}}$ action also prevents an adversary from impersonating parties by creating new groups, as authentication using the compromised identity key will no longer be accepted by other parties.

We can similarly construct a freshness condition for pairwise protocols with PCS-SIG, as sketched in Section C.7.

5.5. Using RSIG in MLS Draft-11 and Signal

In this section, we describe some of the concrete, practical implications of using our proposed PCS-SIG secure signature scheme RSIG with MLS or Signal. Using RSIG in MLS Draft-11 and Signal improves security properties as described above, but also requires additional operations. For Signal, the key to update is the Identity Key using RSIG with the XEdDSA signature scheme. For MLS, where authentication is not well-defined, we assume a single identity signature key is used across all groups. Here we consider computational and network costs associated with RSIG use in both cases.

Computational and Network Cost Overhead. For a party A a PCS-SIG Update requires the following operations:

1) A performs the Update operation, including the creation of a signature σ over a freshly generated public key of length $|pk|$ bytes, incurring a computational cost c_σ.
2) A distributes the resulting update message m_u of length $|m_u| = |\sigma| + |pk|$ bytes to its contacts. The network overhead here depends on the base protocol. In most cases, m_u will be uploaded to an authentication service. In any case, all of A's contacts must be notified about the update. Each contact must then downloads that payload, either from the authentication service or directly from A, and process it, resulting in at least one signature verification operation.
3) A refreshes any prepublished key material that was signed with the old signature key. The overhead again depends on the base protocol and the number of prepublished keys n_{pre}. For MLS, all unused key packages should be replaced. For Signal, only the unused signed prekeys are affected. In both cases, n_{pre} new prepublished keys must be signed, resulting in $n_{\text{pre}} \cdot c_\sigma$ computational overhead and the upload of $n_{\text{pre}} \cdot |PPK|$ bytes of payload, where $|PPK|$ is the size of the respective prepublished key material.

Thus, approximately $|\sigma| + |pk| + n_{\text{pre}} \cdot |PPK|$ of payload must be sent on the network and A has an estimated $(n_{\text{pre}} + 1) \cdot c_\sigma$ computational overhead associated to use of RSIG. We do not include signature length as additional cost in the prekey bundle, as the update does not add to the typical length, or adding computational cost in generation.

Update Frequency and Key Expiration. Both cost and security depend on when and how often an Update operation is performed. In both protocols, prepublished keys are marked with expiration dates, to enforce regular key refreshment. This expiration date implicitly determines the update frequency and indicates the protocol's expectations regarding acceptable recovery windows from compromise for these keys.

From a security standpoint, attaching an expiration date to RSIG keys limits the time that an adversary can circumvent the security of RSIG by simply preventing the victim from distributing an Update. If an expiration date is added to the key, other parties will know not to accept any new messages from that party. See Appendix C.5 for a more detailed discussion of how key expiration affects PCS-SIG security.

RSIG overhead cost can be reduced by aligning the regular update of existing key material necessitated by expiration dates with Update operations. The payload m_u must still be created and published, but due to the expiration date, other parties are aware that the old key has a limited lifetime and will query the authentication service to download the new key, thus removing the necessity for a broadcast by the updating party. Since the prepublished key material must be refreshed anyway, there is no additional exchange of prepublished key material due to the Update under this method.

6. Conclusions

We perform the first cross-group analysis of healing in secure messaging, and formally show that achieving post-compromise security by updating key material at a local-level leaves a significant window of opportunity for an adversary and fails to provide global-level PCS guarantees. Messaging protocols such as ART, TreeKEM, and MLS Draft-11 fail to achieve PCS in concurrent and future groups, due to a lack of synchronized authenticity updates.

We show that these weaknesses can be mitigated by updating long-term signature keys and propose PCS-SIG as a security notion for post-compromise secure signatures. We provide a PCS-SIG provably secure construction for ratcheting digital signatures (RSIG) based on EUF-CMA signatures.

We discussed our work in the MLS working group, which has already prompted changes to the draft that allow applications to make the decision on allocation and ratcheting of the signature keys, and signature keys may be group specific (delegated by an identity-level signature key) or global. The MLS architecture document is planned to contain caveats on security guarantees provided for various application-related decisions, explicitly referring to our work. A concrete implementation of PCS-SIG is currently being explored by Wire.

The use of PCS signatures as opposed to "regular" signature schemes brings potential difficulties in deployment such as key lifetime management. We leave these issues to future work and refer the interested reader to real-world protocols using similar constructions as the ones mentioned in Section C.3. Our work shows that the emerging formal/cryptographic models for group messaging can be strengthened by explicitly considering cross-group effects of healing.

References

[1] DHS/CBP/PIA-008 – Border Searches of Electronic Devices, May 2019. https://www.dhs.gov/publication/border-searches-electronic-devices.

[2] M. Abdalla, F. Benhamouda, and D. Pointcheval. On the tightness of forward-secure signature reductions. Cryptology ePrint Archive, Report 2017/746, 2017. https://eprint.iacr.org/2017/746.

[3] J. Alwen, M. Capretto, M. Cueto, C. Kamath, K. Klein, I. Markov, G. Pascual-Perez, K. Pietrzak, M. Walter, and M. Yeo. Keep the dirt: Tainted treekem, adaptively and actively secure continuous group key agreement. Cryptology ePrint Archive, Report 2019/1489, 2019. https://eprint.iacr.org/2019/1489.

[4] J. Alwen, S. Coretti, and Y. Dodis. The double ratchet: Security notions, proofs, and modularization for the signal protocol. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, Lecture Notes in Computer Science. Springer, 2019.

[5] J. Alwen, S. Coretti, Y. Dodis, and Y. Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. Cryptology ePrint Archive, Report 2019/1189, 2019. https://eprint.iacr.org/2019/1189.

[6] J. Alwen, S. Coretti, D. Jost, and M. Mularczyk. Continuous group key agreement with active security. Cryptology ePrint Archive, Report 2020/752, 2020. https://eprint.iacr.org/2020/752.

[7] R. Barnes, B. Beurdouche, J. Millican, E. Omara, K. Cohn-Gordon, and R. Robert. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-11, IETF Secretariat, December 2020.

[8] M. Bellare and S. K. Miner. A forward-secure digital signature scheme. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 431–448. Springer, Heidelberg, Aug. 1999.

[9] M. Bellare and S. K. Miner. A forward-secure digital signature scheme. In *Advances in Cryptology - CRYPTO '99 Proceedings*, pages 431–448, 1999.

[10] B. Beurdouche. Re: [MLS] long term identity key rotation suggestion. IETF Mail Archive, https://mailarchive.ietf.org/arch/msg/mls/VwzrNH1oMtb8hvwUS9KWqGjhVuk/, Nov. 2019.

[11] K. Bhargavan, R. Barnes, and E. Rescorla. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups, May 2018. Published at https://mailarchive.ietf.org/arch/msg/mls/e3ZKNzPC7Gxrm3Wf0q96dsLZoD8.

[12] N. Borisov, I. Goldberg, and E. A. Brewer. Off-the-record communication, or, why not to use PGP. In V. Atluri, P. F. Syverson, and S. D. C. di Vimercati, editors, *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES 2004, Washington, DC, USA, October 28, 2004*, pages 77–84. ACM, 2004.

[13] X. Boyen, H. Shacham, E. Shen, and B. Waters. Forward-secure signatures with untrusted update. In A. Juels, R. N. Wright, and S. Vimercati, editors, *ACM CCS 06*, pages 191–200. ACM Press, Oct. / Nov. 2006.

[14] C. Brzuska, A. Delignat-Lavaud, C. Fournet, K. Kohbrok, and M. Kohlweiss. State separation for code-based game-playing proofs. LNCS, pages 222–249. Springer, Heidelberg, Dec. 2018.

[15] C. Cimpanu. FBI: Nation-state actors have breached two US municipalities, January 2020. https://www.zdnet.com/article/fbi-nation-state-actors-have-breached-two-us-municipalities/.

[16] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the signal messaging protocol. Cryptology ePrint Archive, Report 2016/1013, 2016. http://eprint.iacr.org/2016/1013.

[17] 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 (IEEE EuroS&P)*, pages 451–466, April 2017.

[18] K. Cohn-Gordon, C. Cremers, and L. Garratt. Post-compromise security. Cryptology ePrint Archive, Report 2016/221, 2016. http://eprint.iacr.org/2016/221.

[19] K. Cohn-Gordon, C. Cremers, L. Garratt, J. Millican, and K. Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. Cryptology ePrint Archive, Report 2017/666, 2017. http://eprint.iacr.org/2017/666.

[20] E. Cronin, S. Jamin, T. Malkin, and P. D. McDaniel. On the performance, feasibility, and use of forward-secure signatures. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *ACM CCS 03*, pages 131–144. ACM Press, Oct. 2003.

[21] Z. Doffman. China's Hackers Accused Of 'Mass-Scale Espionage' Attack On Global Cellular Networks, January 2019. https://www.forbes.com/sites/zakdoffman/2019/06/25/chinese-government-suspected-of-major-hack-on-10-global-phone-companies-reports/.

[22] B. Dowling and B. Hale. There can be no compromise: The necessity of ratcheted authentication in secure messaging. *IACR Cryptol. ePrint Arch.*, 2020.

[23] F. B. Durak and S. Vaudenay. Bidirectional asynchronous ratcheted key agreement with linear complexity. In N. Attrapadung and T. Yagi, editors, *Advances in Information and Computer Security - 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28-30, 2019, Proceedings*, volume 11689 of *Lecture Notes in Computer Science*, pages 343–362. Springer, 2019.

[24] S. Gallagher. Researchers discover state actor's mobile malware efforts because of YOLO OPSEC, January 2019. https://arstechnica.com/information-technology/2019/01/researchers-discover-state-actors-mobile-malware-efforts-because-of-yolo-opsec/.

[25] G. Itkis and L. Reyzin. Forward-secure signatures with optimal signing and verifying. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 332–354. Springer, Heidelberg, Aug. 2001.

[26] D. Jost, U. Maurer, and M. Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Y. Ishai and V. Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 159–188. Springer, 2019.

[27] D. Jost, U. Maurer, and M. Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Y. Ishai and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 159–188, Cham, 2019. Springer International Publishing.

[28] D. Jost, U. Maurer, and M. Mularczyk. A unified and composable take on ratcheting. In D. Hofheinz and A. Rosen, editors, *Theory of Cryptography*, pages 180–210, Cham, 2019. Springer International Publishing.

[29] Keybase. Meet your sigchain (and everyone else's). Technical report, 2020.

[30] H. Krawczyk. Simple forward-secure signatures from any signature scheme. In S. Jajodia and P. Samarati, editors, *ACM CCS 00*, pages 108–115. ACM Press, Nov. 2000.

[31] B. Libert, J.-J. Quisquater, and M. Yung. Forward-secure signatures in untrusted update environments: efficient and generic constructions. In *ACM CCS 07*, pages 266–275. ACM Press, Oct. 2007.

[32] T. Malkin, D. Micciancio, and S. K. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 400–417. Springer, Heidelberg, Apr. / May 2002.

[33] M. Marlinspike and T. Perrin. The Signal Protocol. Technical report, November 2016.

[34] H. Osborne and S. Cutler. Chinese border guards put secret surveillance app on tourists' phones, July 2019. https://www.theguardian.com/world/2019/jul/02/chinese-border-guards-surveillance-app-tourists-phones.

[35] 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 2018, London, United Kingdom, April 24-26, 2018*. IEEE, 2018.

[36] R. Steinfeld, J. Pieprzyk, and H. Wang. How to strengthen any weakly unforgeable signature into a strongly unforgeable signature. In M. Abe, editor, *Topics in Cryptology – CT-RSA 2007*, pages 357–371, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[37] I. Teranishi, T. Oyama, and W. Ogata. General conversion for obtaining strongly existentially unforgeable signatures. In R. Barua and T. Lange, editors, *Progress in Cryptology - INDOCRYPT 2006*, pages 191–205, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[38] E. Thomas. Sydney airport seizure of phone and laptop 'alarming', say privacy groups, August 2018. https://www.theguardian.com/world/2018/aug/25/sydney-airport-seizure-of-phone-and-laptop-alarming-say-privacy-groups.

Appendix A. Groups Using Sender Keys

To improve the scaling behaviour for larger groups, Signal offers another mode called *sender keys*. In this case, when starting a group, each party A generates their own "sender" key k_A, which is a symmetric key that is used in a one-to-many fashion within the context of the group. A then transmits this sender key k_A to the other group members over the pairwise channels. When A wants to send a message to the group, they encrypt it under their own sender key and broadcast the result to the group. The other parties use the sender's key to decrypt the message. A party will generate a sender key for each group they are in. This mechanism is very efficient for large groups, but does not provide PCS by itself: if the adversary learns the state of A, they learn the symmetric sender keys of that party and its peers, and all future messages can be decrypted or forged.

One can reintroduce a form of PCS onto this mechanism by frequently generating new sender keys, and sending them over the pairwise channels and updating those with asymmetric keys at the same time, at the cost of efficiency.

Remark on "sender keys with PCS key updates". For this version of our work we are not explicitly considering the "sender keys with PCS updates" variant of the Signal protocol, in which Alice can update her group-specific sender key over the pairwise channels at some intervals. We do note that, for a passive adversary, the update mechanism for Signal's sender keys protocol heals the future messages *sent by Alice* to the updated group, but still allows a passive adversary to eavesdrop on (i) messages *sent by anyone else in the group* to which Alice just sent an update, and (ii) messages sent or received by Alice in other groups. In that sense, combining sender keys with key updates has weaker PCS guarantees than ART or MLS: in sender keys, a group of size N uses N sender keys, and an update by Alice heals only one of them. In this case, a group in which one member was compromised, the group is only healed with respect to a passive adversary once all members of the group have updated their sender key. This is in contrast to groups based on pairwise PCS channels or ART or MLS Draft-11, in which groups can be healed by only using updates from the compromised party.

Appendix B. Security Guarantees

As shown in Section 3 with the example of two specific scenarios, there are significant differences between the PCS guarantees achieved by ART and MLS Draft-11 and those achieved by groups based on pairwise channels. In this section, we give an intuition how an RSIG scheme can be used to close this gap. Here we showcase how a new Scenario 3 using Construction 1 with ART or MLS Draft-11 provides PCS guarantees beyond what is currently provided.

To that end, we introduce a protocol π that behaves like MLS Draft-11, but replaces the existing signature scheme with the construction defined in Construction 1, instantiated with an EUF-CMA secure signature scheme such as EdDSA. Additionally, the protocol enforces a policy that mandates parallelized (i.e. epoch) encryption updates to ensure complete confidentiality healing in all current groups. These epochs include an update of the signature key.

In the following discussion we step through the scenarios of Section 3 for Approach 1 and Approach 2 using π.

Effects of Approach π in Scenario 1: This fixes the problems exposed in Approach 2 of Scenario 1, and also yields PCS for new groups and conversations. Furthermore, it prevents the problems exposed as a corollary under Approach 2; namely, the adversary gains no advantage in impersonation by dynamically creating new users.

Effects of Approach π in Scenario 2: The ratcheting signature of π allows the scenario to achieve PCS. If Alice has previously communicated with Bob, then the communication is healed due to epoch encryption updates in the same way as in Approach 2. If Alice has not previously communicated with Bob, then the ratcheting signature prevents an active adversary from impersonating Alice in any conversation, even if no previous conversation took place. Note that here the authentication of Alice is globally healed, whereas in the pairwise approach authentication heals only with respect to existing communication partners.

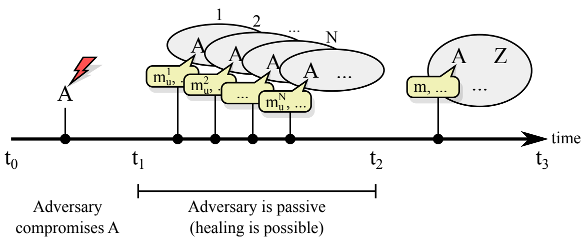


Figure 5: Scenario 3, where Z is not in groups $[1, \dots, N]$.

Below we specifically consider the context of new users as a separate scenario, Scenario 3, which showcases the additional PCS guarantees that π achieves.

Scenario 3. A gets compromised before t_1. She then sends an update message to all her groups in t_1. Subsequently, she starts a new group that includes Z at t_2, where there exist no previous communication between A and Z (meaning that A and Z do not share membership in any group, including pairwise communication). This is depicted in Figure 5.

Effects of Approach 1 *in Scenario 3:* Pairwise channels do not give PCS guarantees in this case, as even after updating all existing channels an adversary can still impersonate A towards Z. The underlying reason is that no previous communication channels exist among the new group members.

Effects of Approach 2 *in Scenario 3:* Neither MLS Draft-11 nor ART achieve PCS guarantees here as no updated key material is shared with Z.

Effects of Approach π in Scenario 3: The ratcheting signature of π allows it to achieve PCS in Scenario 3 in the same way as in Scenario 2. The updated signature key prevents the adversary from impersonating A towards Z and others and thus allows A to achieve full (authenticity/confidentiality) PCS once the first encryption update is sent.

Appendix C. PCS-signatures Model Discussion

C.1. Experiment Details

Throughout the experiment, a sequence number sqn increments according to the number of PCS-SIG.Update queries the adversary makes, denoting the current epoch. This is used for global ordering. Note, that instead of providing an "initialization phase" for the experiment, we rely on variables being assigned a default value $\perp$ for single-value variables or $\emptyset$ for sets, and provide the oracles SignerGen and VerifierGen for explicit initialization of the initial key material. This is to conform with the style and notation introduced in [14], which aims at easing the modelling of larger, composed protocols using PCS-SIG.

- PCS-SIG.SignerGen may be called by the adversary to initialize the signer key. It can only be called once.

- PCS-SIG.VerifierGen may be called by the adversary to initialize the verifier key with the value of the initial signer key. It can only be called once and only after the signer key was initialized.
- PCS-SIG.Update may be called by the adversary to operate on the current signer key. It calls $\mathrm{RSIG.Update}$ with the private key of the current epoch and assigns the resulting public and private key to be the public and private key of the next epoch. It increments the epoch counter sqn and record the new epoch number and the message in the UpdateList. Finally, it returns the new epoch key and the update message to the adversary.
- PCS-SIG.Sign may be called by the adversary on any message of their choosing. The signature, message, and current public key of the signer are recorded and the signature is returned to the adversary.
- PCS-SIG.Corrupt may be called by the adversary on any epoch number of their choosing for which a key has been assigned. Otherwise, the current sequence number is recorded and the secret key is returned to the adversary.
- PCS-SIG.RcvUpdate may be called by the adversary on any update message (generated by PCS-SIG.Update or forged). The oracle processes the update message under the currently recorded public key of the signer and checks if the output is a valid public key. If so, the verifier side updates its recorded public key for the signer and adds the update message to its record list, returning the updated public key to the adversary. Note that this requires only that a valid public key is output from $\mathrm{RSIG.RcvUpdate}$, not that it matches the actual updated signer public key. Consequently, an adversary may forge an updated that is accepted, resulting in diverging signer and verifier public keys.
- PCS-SIG.Verify may be called by the adversary on any message and signature pair (generated by PCS-SIG.Sign or forged). The oracle processes the signature verification and allows a win to the adversary if all of the following hold:
  - An update message has been both correctly generated and received following any and all corruption queries.
  - The message was not previously submitted to the PCS-SIG.Sign oracle. Note that this requirement fails if the message or public key do not match (i.e. existential unforgeability inclusive of public key).
  - The verification processes correctly.

  If all of the above hold, the adversary wins. Note that the first clause of the conjunction requires that m was both correctly generated and processed, since $m \in \mathsf{RcvdUpdateList}$ implies that the current verifier key was originally generated by the signer.

C.2. Composability

We chose to model the security of RSIG schemes as an indistinguishability game, as opposed to using the "forgery submission" style commonly used for unforgability security notions. This is for reasons of composability both as a building block, for proofs dealing with protocols building on a PCS-SIG secure RSIG, as well as for the security proof of our own RSIG construction, which we model as multiple instances of an EUF-CMA-secure signature scheme.

C.3. Existing Application of PCS-signatures

While we provide the first formal definition for PCS-signatures, using signature keys to sign other signature keys is a common design pattern in real world protocols. For example, in DNSSEC and the HTTPS PKI, signature keys are used to sign other signature keys (as well as additional information) in a hierarchical structure. The platform Keybase uses "SigChains" [29] to manage user account information, which is conceptually close to RSIGs, but more complex as not only signature keys are included in the chain.

C.4. SUF-CMA vs. EUF-CMA

In the security experiment, if every corruption event was followed by an "honest" update, the PCS-SIG.Verify oracle returns b if the adversary manages to submit an existential forgery to the oracle. We chose to make the requirement an existential forgery so that it is possible to prove constructions secure, where the underlying signature scheme is only EUF-CMA secure (as shown in Section 5.2). It is, however, possible to strengthen PCS-SIG by requiring the adversary to submit a strong forgery to $\mathsf{PCS\text{-}SIG.Verify}$, i.e. the adversary has to submit a pair (m, σ), where σ was not the output of $\mathsf{PCS\text{-}SIG.Sign}(m)$ (with matching public keys between the two oracle queries). This would enable the use of $\mathsf{PCS\text{-}SIG}$ in protocols that require SUF-CMA (Strong Unforgability under Chosen Message Attacks) secure signatures (see [36], [37] for further context on EUF-CMA and SUF-CMA). Moreover, we believe a more flexible security notion is possible, where different signature schemes can be used: one to sign updates and one to sign messages. The former would only have to be EUF-CMA secure, while the latter might give stronger guarantees such as SUF-CMA. We leave further exploration of a more flexible and powerful $\mathsf{PCS\text{-}SIG}$ notion to future work.

C.5. Key Expiration

A crucial part of gaining PCS-Signature security is the expiration of keys. In the PCS-SIG game, we model this by the verifier replacing their current public key with a new one upon receiving a valid update. After that update, the verifier exclusively uses the new public key for verification. An obvious attack would be for the adversary to simply withhold any update and thus prevent the verifier from ever gaining security after a compromise of the signer. There are several ways to approach this problem. One solution is adding timestamps to public keys to indicate their validity period, as is done by both DNSSEC and the HTTPS PKI. The notion of actual time and its secure synchronization between signer and verifier would render the security model significantly more complex. However, given the simplicity of our model, we believe that it would be relatively straight-forward to implement such an expiration mechanism by composing PCS-SIG with a secure time-synchronization protocol. Another way to expire keys is by public revocation. This is implemented by the HTTPS PKI using revocation lists. However, this only shifts the problem, as the adversary can withhold updates to the revocation list from the receiver in the same way as with key updates.

C.6. Deniability

Notably, PCS-signatures can be used to alleviate a potential drawback of static signatures: it is possible to use a similar mechanism as the OTR protocol [12] to explicitly reveal old authentication keys. For PCS-signatures, explicitly leaking old private signing keys can be used to achieve stronger deniability properties.

C.7. Freshness for Signal+PCS-SIG

The additional guarantees resulting in the addition of PCS-SIG to Signal are analogous to those in case of MLS in Definition 8, as a $\mathit{AuthUpd}_{\mathit{passive}}$ action means that we don't require the adversary be passive during a subsequent GrpUpd action.

Definition 9 ($\mathit{Fresh}_{\mathit{Signal}+\mathit{PCS\text{-}SIG}}$)**.** *For session identifier sid, we define $\mathit{Fresh}_{\mathit{Signal}}(\mathit{sid})$ as:*

$$\begin{aligned}
&\big(\mathit{Fresh}_{\mathit{base}}(\mathit{sid}) \land \neg\exists X \ : \ \mathit{FullCompromise}(X)\big) \lor \\
&\big(\forall X \in \mathit{sid}.G \ : \\
&\exists \mathit{sid}' \ : \ \mathit{sid}'.\mathit{actor} = \mathit{sid}.\mathit{actor} \land \mathit{GrpUpd}_{\mathit{passive}}(X, \mathit{sid}')\big) \lor \\
&\big(\forall X \in \mathit{sid}.G \ : \ \mathit{AuthUpd}_{\mathit{passive}}(X, \mathit{sid}.\mathit{actor}) < \\
&\exists \mathit{sid}' \ : \ \mathit{sid}'.\mathit{actor} = \mathit{sid}.\mathit{actor} \land \mathit{GrpUpd}_{\mathit{active}}(X, \mathit{sid}')\big) \ .
\end{aligned}$$

In the same way as for MLS, the adversary can't create new groups after an $\mathit{AuthUpd}_{\mathit{passive}}$ action, in Signal, the adversary can't impersonate a compromised party to other parties without an existing one-to-one connection. Again, the reason is that other parties won't accept an updated identity key to authenticate the previously compromised party.

Appendix D. Security Proof

In our proof, we rely on the notation and some of the proof techniques of the state-separating proof (SSP) framework introduced in [14], which we will briefly state here.

SSP Notation and Definitions. In traditional game-playing proofs, the adversary has access to a set of named oracles, which operate on some shared private state. An oracle is an algorithm defined using pseudocode, taking some input on which it operates, and producing an output in the end. We can thus view an entire cryptographic security game G as a set of oracles Ω and some state that these oracles operate on. In the SSP framework, this is called a *package*. An adversary $\mathcal{A}$ interacting with a game G is denoted as follows:

$$\mathcal{A} \circ \mathsf{G} \ .$$

We call the set of names of the oracles defined in G the *output interface* of G, denoted by $\mathsf{out}(\mathsf{G}) = \{\mathsf{O}_1, \mathsf{O}_2, \dots\}$, where $\mathsf{O}_1, \mathsf{O}_2, \dots$ are the names of the oracles $\mathsf{O} \in \Omega$. For convenience, we use the name of an oracle O to both denote the oracle itself and its name, denoting the difference explicitly where applicable. We also say that the oracles $\mathsf{O} \in \mathsf{out}(\mathsf{G})$ are *provided* by G.

Oracles provided by a package P can query one or more oracles that are in turn provided by another package P'. Note, that oracles can't query other oracles provided by the *same* package. We call packages P, P' *sequentially composed* and denote this with $\mathsf{P} \circ \mathsf{P}'$. The *input interface* of a package P (denoted $\mathsf{in}(\mathsf{P})$) is the set of oracle names that oracles provided by P make queries to. Reductions as commonly used in traditional game-playing proofs make use of the same concept.

Given a security notion G, an algorithm instantiating a reduction R is defined in terms of an adversary $\mathcal{A}$ against G and an assumption G' (which is in turn a security game). The goal of the reduction is to reduce the security of G to that of G'. We use the term "reduction" to denote both the relation between G and G', as well as its concrete instantiation R. R leverages $\mathcal{A}$ to break the security of G'. When running $\mathcal{A}$, the reduction has to provide a set of oracles for $\mathcal{A}$ to interact with. We can thus see R as a package. From the point of view of the adversary, the reduction is just like a security game, which is why we can denote an adversary interacting with a reduction with $\mathcal{A} \circ \mathsf{R}$. The oracles provided by R in turn interact with the assumption by querying the oracles provided by G', i.e. R and G' are sequentially composed. The complete reduction can thus be written as $\mathcal{A} \circ \mathsf{R} \circ \mathsf{G}'$.

Relations between packages. We use $\mathsf{G} \equiv \mathsf{G}'$ to denote perfect equivalence between packages. Let G^b be a game parameterized by $b \in \{0, 1\}$. Then we use $\mathsf{G}^0 \overset{\epsilon_{\mathsf{G}}(\mathcal{A})}{\approx} \mathsf{G}^1$ to denote that for all adversaries $\mathcal{A}$ against G, we have that G^0 is indistinguishable from G^1 up to some advantage function $\epsilon(\mathcal{A})$.

SSP Proofs. Besides providing a notation for definitions, the SSP framework is also meant to help with some types of reduction proof. Continuing the example above, the next step in the proof would be to show simulation correctness, i.e. to show that $\mathsf{R} \circ \mathsf{G}'$ indeed simulates the game G correctly to $\mathcal{A}$. In terms of packages, this means proving that the oracles provided by $\mathsf{R} \circ \mathsf{G}'$ are perfectly equivalent to those provided by G. We denote perfect equivalence using the symbol $\equiv$. We thus have to show that

$$\mathsf{G} \equiv \mathsf{R} \circ \mathsf{G}' \ .$$

To continue with our example, let G be an indistinguishability game with a distinguishing bit b. We use G^b to denote that G is parameterized with b. Similarly, let G' be an indistinguishability game with a distinguishing bit b'. This means proving simulation correctness means proving

$$\mathsf{G}^b \equiv \mathsf{R} \circ \mathsf{G}'^{b'} \ ,$$

where $b = b'$.

In the SSP framework, indistinguishability statements are denoted as follows. If we have that for all adversaries $\mathcal{A}'$ against our assumption G', we have that G'^0 is indistinguishable from G'^1 up to some advantage function $\epsilon(\mathcal{A}')$, we write

$$\mathsf{G}'^0 \overset{\epsilon_{\mathsf{G}'}(\mathcal{A}')}{\approx} \mathsf{G}'^1 \ .$$

Given that for some adversary $\mathcal{A}$ against G, $\mathcal{A} \circ \mathsf{R}$ is technically just another adversary and thus falls under the "for all adversaries" quantification, as a corollary of our assumption, we get for all adversaries

$$\mathcal{A} \circ \mathsf{R} \circ \mathsf{G}'^0 \overset{\epsilon_{\mathsf{G}'}(\mathcal{A} \circ \mathsf{R})}{\approx} \mathcal{A} \circ \mathsf{R} \circ \mathsf{G}'^1 \ .$$

To obtain a full proof, simply have to connect the simulation correctness proof and the assumption:

$$\begin{aligned} \mathsf{G}^0 &\equiv \mathsf{R} \circ \mathsf{G}'^0 \\ &\equiv \mathsf{G}^1 \ , \end{aligned}$$

which by the transitivity of $\approx$ and $\equiv$ gives us

$$\mathsf{G}^0 \overset{\epsilon_{\mathsf{G}'}(\mathcal{A}\circ\mathsf{R})}{\approx} \mathsf{G}^1 .$$

Multi-Instance in SSP. Another feature of SSP we use in our proof is a simple conversion of an individual primitive (EUF-CMA in our case) into its multi-instance variant. For the case of EUF-CMA, this means that instead of interacting with a single keypair, the adversary instead interacts with a number of keypairs. The adversary wins if it manages to create a forgery for any one of those keypairs. The conversion from single-instance to multi-instance is done using another kind of composition called *parallel composition*. For the technical details, see Lemma 28 in [14].

D.1. Assumptions

Before we state our main security theorem, we start by stating our standard assumption. We then reduce corruptible EUF-CMA (CEUF-CMA) to standard EUF-CMA and lift CEUF-CMA to a multi-instance version of itself (MI-CEUF-CMA) using a standard hybrid argument.

Finally, we reduce PCS-SIG to MI-CEUF-CMA.

Definition 10 (Standard EUF-CMA Assumption)**.** *Let ν be a digital signature scheme. Then we say ν is EUF-CMA secure with adversarial advantage function $\epsilon_{\mathsf{EUF\text{-}CMA}}(\cdot)$, if for all adversaries $\mathcal{A}$ we have that*

$$\mathsf{EUF\text{-}CMA}^{0,\nu} \overset{\epsilon_{\mathsf{EUF\text{-}CMA}}(\mathcal{A})}{\approx} \mathsf{EUF\text{-}CMA}^{1,\nu},$$

where for $b \in \{0,1\}$ the oracles provided by $\mathsf{EUF\text{-}CMA}^{b,\nu}$ are described in Figure 6:

```
Gen()
$pk, sk \leftarrow\$ \nu.\mathrm{Gen}()$
return $pk$

Sign($m$)
assert $sk \neq \bot$
$\sigma \leftarrow \nu.\mathrm{Sign}(sk, m)$
$\mathsf{MsgList} \leftarrow \mathsf{MsgList} \cup \{m\}$
return $\sigma$

Verify($m, \sigma$)
assert $pk \neq \bot$
$\mathrm{verify} \leftarrow \nu.\mathrm{Verify}(pk, m, \sigma)$
if $m \notin \mathsf{MsgList} \wedge \mathrm{verify} = \mathit{true}$ then
    return $b$
return verify
```

Figure 6: Description of CEUF-CMA oracles.

We proceed with the definition of corruptible EUF-CMA, where the only difference between the traditional EUF-CMA is that an adversary can decide to use the Corrupt oracle to obtain the secret key. However, this effectively disables the Verify oracle, preventing the adversary from learning anything about b.

Definition 11 (Corruptible EUF-CMA)**.** *Let ν be a digital signature scheme. Then we say ν is CEUF-CMA secure with adversarial advantage function $\epsilon_{\mathsf{CEUF\text{-}CMA}}(\cdot)$, if for all adversaries $\mathcal{A}$ we have that*

$$\mathsf{CEUF\text{-}CMA}^{0,\nu} \overset{\epsilon_{\mathsf{CEUF\text{-}CMA}}(\mathcal{A})}{\approx} \mathsf{CEUF\text{-}CMA}^{1,\nu},$$

where for $b \in \{0,1\}$ the oracles provided by $\mathsf{CEUF\text{-}CMA}^{b,\nu}$ are described in Figure 7.

Theorem 1. *Let ν be an $\epsilon_{\mathsf{EUF\text{-}CMA}}$-EUF-CMA secure digital signature scheme and $b \in \{0,1\}$. Then for all adversaries $\mathcal{A}$ against $\mathsf{CEUF\text{-}CMA}^{b,\nu}$ there exists a reduction $\mathsf{R}_{\mathsf{CEUF\text{-}CMA}}$ such that*

$$\epsilon_{\mathsf{CEUF\text{-}CMA}}(\mathcal{A}) \leq \epsilon_{\mathsf{EUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R}_{\mathsf{CEUF\text{-}CMA}}).$$

```
Gen()
    assert $sk = \bot$
    $pk, sk \leftarrow\$ \nu.\mathrm{Gen}()$
    return $pk$

Corrupt()
    assert $sk \neq \bot$
    $\mathsf{C} \leftarrow 1$
    return $sk$

Sign($m$)
    assert $sk \neq \bot$
    $\sigma \leftarrow \nu.\mathrm{Sign}(sk, m)$
    $\mathsf{MsgList} \leftarrow \mathsf{MsgList} \cup \{m\}$
    return $\sigma$

Verify($m, \sigma$)
    assert $pk \neq \bot$
    $\mathrm{verify} \leftarrow \nu.\mathrm{Verify}(pk, m, \sigma)$
    if $m \notin \mathsf{MsgList} \wedge \mathrm{verify} = \mathit{true} \wedge \mathsf{C} \neq 1$ then
        return $b$
    return $v$
```

Figure 7: Description of CEUF-CMA oracles.

Proof. Let $\mathsf{R}_{\mathsf{CEUF\text{-}CMA}}$ be a package that forwards all queries to the EUF-CMA game and returns a fixed random value upon any query to the Corrupt oracle. Let $\mathcal{A}$ be an adversary against CEUF-CMA. Since $\mathcal{A}$ can not gain any information about b besides receiving it directly as a return value from Verify, calling the Corrupt oracle means that it has either received b before that point in time, or the best it can do is guess b at random regardless of the return value of any queries to Corrupt. Thus, the advantage of $\mathcal{A}$ against CEUF-CMA is upper bounded by its advantage against $\mathsf{R}_{\mathsf{CEUF\text{-}CMA}} \circ \mathsf{EUF\text{-}CMA}$. □

Definition 12 (Multi-Instance Corruptible EUF-CMA)**.** *Let ν be a digital signature scheme. Then we say ν is MI-CEUF-CMA secure with adversarial advantage function $\epsilon_{\mathsf{MI\text{-}CEUF\text{-}CMA}}(\cdot)$, if for all adversaries $\mathcal{A}$ we have that*

$$\mathsf{MI\text{-}CEUF\text{-}CMA}^{0,\nu} \overset{\epsilon_{\mathsf{MI\text{-}CEUF\text{-}CMA}}(\mathcal{A})}{\approx} \mathsf{MI\text{-}CEUF\text{-}CMA}^{1,\nu},$$

where $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu} := \prod_{i=1}^{n} \mathsf{CEUF\text{-}CMA}_i^{b,\nu}$.

Note that with $\prod_{i=1}^{n} \mathsf{M}$ we denote the *parallel composition* of n instances of a package M. This means that any package M' interacting with $\prod_{i=1}^{n} \mathsf{M}$ has access to n instances of M. We use an index i to indicate which instance an oracle of M' is interacting with, i.e. $x \leftarrow \mathsf{O}_i$ in the description of an oracle of M' would indicate a call to oracle O of instance i, assigning the return value to the variable x.

Theorem 2 (Multi-Instance Corruptible EUF-CMA)**.** *Let ν be an EUF-CMA-secure digital signature scheme with adversarial advantage function $\epsilon_{\mathsf{EUF\text{-}CMA}}$, and $b \in \{0,1\}$. Then for all adversaries $\mathcal{A}$ against* $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu}$ *we have that there exists a reduction* $\mathsf{R}_{\mathsf{MI\text{-}CEUF\text{-}CMA}}$ *such that*

$$\epsilon_{\mathsf{MI\text{-}CEUF\text{-}CMA}}(\mathcal{A}) \leq n \cdot \epsilon_{\mathsf{CEUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R}_{\mathsf{MI\text{-}CEUF\text{-}CMA}}),$$

where n is the number of $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu}$ *instances.*

Proof. The proof is a simple instantiation of Lemma 28 in [14], essentially a hybrid argument over n $\mathsf{CEUF\text{-}CMA}^{b,\nu}$ packages. The hybrid argument gives us a reduction $\mathsf{R}_{\mathsf{MI\text{-}CEUF\text{-}CMA}}$ such that $\forall \mathcal{A}, \epsilon_{\mathsf{MI\text{-}CEUF\text{-}CMA}}(\mathcal{A}) \leq n \cdot \epsilon_{\mathsf{CEUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R}_{\mathsf{MI\text{-}CEUF\text{-}CMA}})$. Using Theorem 1 concludes the proof. □

Note, that Lemma 28 of [14] defines n as static, i.e. the adversary can not decide how many instances to interact with dynamically. Instead, n is fixed before the proof. While this restricts the reduction to the class of adversaries interacting with n instances, the reduction can be instantiated for any $n \in \mathbb{N}$, thus giving us security against adversaries interacting with any number of instances.

D.2. Main Theorem and Proof

We now state the main theorem and prove it under the assumptions described in the previous section.

Theorem 3 (Construction 1 is PCS-SIG-secure)**.** *Let $\nu = (\mathrm{Gen}, \mathrm{Sign}, \mathrm{Verify})$ be an EUF-CMA-secure signature scheme with adversarial advantage function $\epsilon_{\mathsf{EUF\text{-}CMA}}$, and $\mu = (\mathrm{Gen}, \mathrm{Sign}, \mathrm{Verify}, \mathrm{Update}, \mathrm{RecvUpdate})$ constructed as in Construction 1. Then μ is PCS-SIG-secure with adversarial advantage function $\epsilon_{\mathsf{PCS\text{-}SIG}}$, i.e. there exists a reduction $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ such that we have that for all adversaries $\mathcal{A}$*

$$\epsilon_{\mathsf{PCS\text{-}SIG}}(\mathcal{A}) \leq n \cdot \epsilon_{\mathsf{MI\text{-}CEUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R}_{\mathsf{PCS\text{-}SIG}}),$$

where $n \in \mathbb{N}$ is the number of calls to $\mathsf{PCS\text{-}SIG.Update}$.

Proof. Let μ be the RSIG construction based on the digital signature scheme ν. We define a reduction $\mathsf{R_{PCS\text{-}SIG}}$ as shown in Fig. 8 that interfaces with $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu}$ and that presents the same interface to the adversary as $\mathsf{PCS\text{-}SIG}$.

We then proceed to prove the theorem using two claims. The first is that for $b \in \{0, 1\}$

$$\mathsf{R_{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu} \equiv \mathsf{PCS\text{-}SIG}^{b,\mu},$$

where with $\equiv$ we denote functional equivalence. In the more traditional style of cryptographic proof, this corresponds to showing that our reduction simulates the security game correctly for the adversary. The second is that

$$\begin{aligned}&\mathsf{R_{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CEUF\text{-}CMA}^{0,\nu}\\ &\overset{\epsilon_{\mathsf{MI\text{-}CEUF\text{-}CMA}}(\mathcal{A}\circ\mathsf{R_{PCS\text{-}SIG}})}{\approx} \mathsf{R_{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CEUF\text{-}CMA}^{1,\nu}.\end{aligned}$$

Combining both claims allows us to state that

$$\begin{aligned}\mathsf{PCS\text{-}SIG}^{0,\mu} &\equiv \mathsf{R_{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CEUF\text{-}CMA}^{0,\nu}\\ &\overset{\epsilon_{\mathsf{MI\text{-}CEUF\text{-}CMA}}(\mathcal{A}\circ\mathsf{R_{PCS\text{-}SIG}})}{\approx} \mathsf{R_{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CEUF\text{-}CMA}^{1,\nu}\\ &\equiv \mathsf{PCS\text{-}SIG}^{1,\mu},\end{aligned}$$

which proves the theorem.

We now state our claims formally, followed by proofs.

Claim 1 (Functional Equivalence of Packages)**.** *Let μ be the RSIG construction based on the digital signature scheme ν, then we have for $b \in bin$*

$$\mathsf{R_{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu} \equiv \mathsf{PCS\text{-}SIG}^{b,\mu}.$$

Proof. To prove equivalence, we will first take a closer look at the reduction algorithm $\mathsf{R_{PCS\text{-}SIG}}$. We construct $\mathsf{R_{PCS\text{-}SIG}}$ such that it behaves exactly as $\mathsf{PCS\text{-}SIG}^{b,\mu}$ when inlining the construction μ, but instead of calling the algorithms of the underlying signature scheme ν, $\mathsf{R_{PCS\text{-}SIG}}$ calls the oracles of $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu}$ of the same name. On the signer side, we use sqn to determine which instance of $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu}$ to interact with. On the verifier side, $\mathsf{R_{PCS\text{-}SIG}}$ only calls the oracle if current $\mathsf{verifier.pk}$ corresponds to some instance of $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu}$. If it does not, it simply calls the Verify algorithm with $\mathsf{verifier.pk}$.

Additionally, if one of the calls to $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu}.\mathsf{Verify}$ returns the distinguishing bit b, $\mathsf{R_{PCS\text{-}SIG}}$ stores it in the variable b_{store} and otherwise carries on just as $\mathsf{PCS\text{-}SIG}^{b,\mu}$, pretending the result was $true$. Finally, the win condition of $\mathsf{R_{PCS\text{-}SIG}}$ is different to that of $\mathsf{PCS\text{-}SIG}^{b,\mu}$ in that the reduction additionally requires b_{store} to be set, i.e. not to be $\bot$.

Proof Overview. We split the proof into two steps. We first argue that, with exception of the win condition, the two games are equivalent by construction of $\mathsf{R_{PCS\text{-}SIG}}$ and $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu}$. Additionally, we compare the oracle code of both games, where we inline oracle calls and merge **if** conditions where applicable to verify that this indeed the case. Secondly, we prove via induction over all oracle calls that we have $b_{\text{stored}} \neq \bot$ whenever the win condition of the security game is true, thus proving that the win conditions of both security game and reduction are indeed equivalent.

Construction Equivalence. The first part of the proof follows from the construction of $\mathsf{R_{PCS\text{-}SIG}}$. The oracles of $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu}$ behave exactly like the underlying signature scheme, except in the case of a forgery, where it outputs the distinguishing bit b. In that case, the reduction stores the bit in b_{store} and otherwise continues operating as if the output were $true$, thus maintaining the same functionality as $\mathsf{PCS\text{-}SIG}^{b,\mu}$. In cases where $\mathsf{R_{PCS\text{-}SIG}}$ can not call oracles on the verifier side, it instead calls the signature scheme directly using the current $\mathsf{verifier.pk}$, maintaining the same functionality. In addition to inlining, we have simplified the code of the reduction by merging **if** branches and simplifying their conditions where applicable.

```
$\mathsf{R}_{\mathsf{PCS\text{-}SIG}}.\mathsf{SignerGen}()$
$pk_1 \leftarrow \mathsf{Gen}_1()$
$\mathsf{sqn} \leftarrow 1$
$\mathbf{return}\ pk_{\mathsf{sqn}}$

$\mathsf{R}_{\mathsf{PCS\text{-}SIG}}.\mathsf{VerifierGen}()$
$\mathbf{assert}\ pk_1 \neq \bot$
$\mathbf{assert}\ \mathsf{verifier.pk} = \bot$
$\mathsf{verifier.pk} \leftarrow pk_1$
$\mathbf{return}\ \mathsf{verifier.pk}$

$\mathsf{R}_{\mathsf{PCS\text{-}SIG}}.\mathsf{Corrupt}(i)$
$\mathbf{assert}\ sk_i \neq \bot$
$sk \leftarrow \mathsf{Corrupt}_i()$
$\mathbf{return}\ sk$

$\mathsf{R}_{\mathsf{PCS\text{-}SIG}}.\mathsf{Update}()$
$pk_{\mathsf{sqn}+1} \leftarrow \mathsf{Gen}_{\mathsf{sqn}+1}()$
$\sigma \leftarrow \mathsf{Sign}_{\mathsf{sqn}}(pk_{\mathsf{sqn}+1} || \text{“update”})$
$m_u \leftarrow (pk_{\mathsf{sqn}+1}, \sigma)$
$\mathsf{sqn} \leftarrow \mathsf{sqn} + 1$
$\mathsf{UpdateList} \leftarrow \mathsf{UpdateList} \cup \{(\mathsf{sqn}, m_u)\}$
$\mathbf{return}\ (pk_{\mathsf{sqn}}, m_u$

$\mathsf{R}_{\mathsf{PCS\text{-}SIG}}.\mathsf{Sign}(m)$
$\sigma \leftarrow \mathsf{Sign}_{\mathsf{sqn}}(m || \text{“msg”})$
$\mathsf{MsgList} \leftarrow \mathsf{MsgList} \cup \{(m, pk_{\mathsf{sqn}})\}$
$\mathbf{return}\ \sigma$

$\mathsf{R}_{\mathsf{PCS\text{-}SIG}}.\mathsf{RcvUpdate}(m_u)$
$\mathbf{assert}\ \mathsf{verifier.pk} \neq \bot$
$(pk', \sigma) \leftarrow m_u$
$\mathbf{if}\ \exists i \in \mathbb{N}$ s.t. $pk_i = \mathsf{verifier.pk}\ \mathbf{then}$
    $\mathrm{verify} \leftarrow \mathsf{Verify}_i((pk' || \text{“update”}), \sigma)$
    $\mathbf{if}\ \mathrm{verify} \in \{0, 1\}\ \mathbf{then}$
        $b_{\mathrm{store}} \leftarrow \mathrm{verify}$
        $\mathrm{verify} \leftarrow \mathit{true}$
$\mathbf{else}$
    $\mathrm{verify} \leftarrow \nu.\mathrm{Verify}(\mathsf{verifier.pk}, (pk' || \text{“update”}), \sigma)$
$\mathbf{if}\ \mathrm{verify} = \mathit{true}\ \mathbf{then}$
    $\mathsf{RcvdUpdateList} \leftarrow \mathsf{RcvdUpdateList} \cup \{m_u\}$
    $\mathsf{verifier.pk} \leftarrow pk'$
$\mathbf{return}\ \mathrm{verify}$

$\mathsf{R}_{\mathsf{PCS\text{-}SIG}}.\mathsf{Verify}(m, \sigma)$
$\mathbf{assert}\ \mathsf{verifier.pk} \neq \bot$
$\mathbf{if}\ \exists i \in \mathbb{N}$ s.t. $pk_i = \mathsf{verifier.pk}\ \mathbf{then}$
    $\mathrm{verify} \leftarrow \mathsf{Verify}_i((m || \text{“msg”}), \sigma)$
    $\mathbf{if}\ \mathrm{verify} \in \{0, 1\}\ \mathbf{then}$
        $b_{\mathrm{store}} \leftarrow \mathrm{verify}$
        $\mathrm{verify} \leftarrow \mathit{true}$
$\mathbf{else}$
    $\mathrm{verify} \leftarrow \nu.\mathrm{Verify}(\mathsf{verifier.pk}, (m || \text{“msg”}), \sigma)$
$\mathbf{if}\ \mathrm{verify} = \mathit{true} \wedge$
    $b_{\mathrm{store}} \neq \bot \wedge$
    $(m, \mathsf{verifier.pk}) \notin \mathsf{MsgList} \wedge$
    $\big(\forall i \in \mathsf{Corrupted}, \exists (j, m_u) \in \mathsf{UpdateList}$ s.t.
        $j > i \wedge$
        $m_u \in \mathsf{RcvdUpdateList}\big)\ \mathbf{then}$
    $\mathbf{return}\ b_{\mathrm{store}}$
$\mathbf{return}\ \mathrm{verify}$
```

Figure 8: Description of the oracles of $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$.

Win Conditions. Before we begin comparing the win conditions, we introduce a few notational conveniences. We call public keys generated by calls to SignerGen or Update “honest”. Formally, this means that by construction of the oracles SignerGen and Update in both the reduction and the security notion, we have that a public key pk is honest if and only if $\exists i \in \mathbb{N}$ s.t. $pk = pk_i$. Also, we use “regular update” to denote the event that the adversary causes an update of verifier.pk by calling RcvUpdate with a message previously output by Update. By construction of Update and RcvUpdate, we know that if a regular update has happened at epoch j, we have that

$$\exists (j, m_u) \in \mathsf{UpdateList} \text{ s.t. } m_u \in \mathsf{RcvdUpdateList},$$

as well as the fact that immediately after the update, $\mathsf{verifier.pk} = pk_j$ i.e. verifier.pk is honest.

In terms of the win condition, there are two differences of note between the reduction and the security notion. The first is that in the reduction we additionally store the bit b in b_{store} if it is returned by a call from $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu}$.Verify. The second is that in the reduction we require in addition to the other win conditions, that $b_{\mathrm{store}} \neq \bot$. We will use the first difference to argue that the second has no impact on the functionality of $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$.Verify, by proving that $b_{\mathrm{store}} \neq \bot$ is implied by the other win conditions.

To argue about the equivalence of the win conditions on the reduction and the security notion, we define a predicate over the joint state of $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ and $\mathsf{PCS\text{-}SIG}^{b,\mu}$. We then go on to prove that the predicate is an invariant, i.e. that if it holds before the call to any oracle, it also holds after the call.

Claim 2 (State Invariant). *For all oracles* O *of* $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ *and all oracles of* $\mathsf{PCS\text{-}SIG}^{b,\mu}$*, if the following predicate over the state holds before a call to* O*, it also holds after the call.*

$$\begin{aligned}\forall i \in \mathbb{N}: \quad & \exists (m || \textit{“msg”}) \in \mathsf{MsgList}_i \iff \\ & \quad \exists pk_i \textit{ s.t. } (m, pk_i) \in \mathsf{MsgList} \land \\ & i \in \mathsf{Corrupted} \iff \mathsf{C}_i = 1 \land \\ & \exists (m_u || \textit{“update”}) \in \mathsf{MsgList}_{i-1} \\ & \quad \iff \exists (i, m_u) \in \mathsf{UpdateList}\end{aligned}$$

Proof. Intuitively, the invariant says that the state of $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ and $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu}$ are consistent. This holds by construction of $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$. □

We also introduce a predicate over the (partial) package state of $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ before and after a call to $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}.\mathsf{RcvUpdate}$, as well as its input m, i.e. a post-condition.

Claim 3 (Post-Condition of $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}.\mathsf{RcvUpdate}$)**.** *Let* $\Sigma :=$ ($\mathsf{verifier.pk}$, $\mathsf{UpdateList}$, $\mathsf{RcvdUpdateList}$, $\mathsf{MsgList}$, $b_{\textit{store}}$, b, $\mathsf{Corrupted}$) *be the (partial) state of* $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ *before the execution of an oracle call to* $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}.\mathsf{RcvUpdate}$ *with input* m_u *and* Σ' *be the same state variables, but after the call. Then we say that the following postcondition* $\mathcal{P}(\Sigma, (pk, \sigma), \Sigma')$ *holds, where* $(pk, \sigma) \leftarrow m_u$*.*

$$\begin{aligned}\exists i \in \mathbb{N} \textit{ s.t.} \quad & (i, (pk, \sigma)) \in \mathsf{UpdateList} \land \\ & \nu.\mathrm{Verify}(\mathsf{verifier.pk}, pk, \sigma) = \textit{true} \implies \\ & \quad ((pk, \sigma) \in \mathsf{RcvdUpdateList}' \land \\ & \quad\quad \mathsf{verifier.pk}' = pk_i) \land \\ \exists i \in \mathbb{N} \textit{ s.t.} \quad & \mathsf{verifier.pk} = pk_i \land \\ & i \notin \mathsf{Corrupted} \land \\ & (pk_i, pk) \notin \mathsf{MsgList} \land \\ & \nu.\mathrm{Verify}(\mathsf{verifier.pk}, pk, \sigma) = \textit{true} \implies \\ & \quad b'_{\textit{store}} = b \ .\end{aligned}$$

Proof. Again, the claim holds by construction of $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$. □

The post-condition ensures two things: 1) that $\mathsf{verifier.pk}$ is honest following a regular update and 2) that if the $\mathsf{verifier.pk}$ was honest (and not corrupted) before the call, it is either also honest after the call, or b_{store} was set. Intuitively, we claim that if the current $\mathsf{verifier.pk}$ is honest, the adversary has to forge a signature to insert their own $\mathsf{verifier.pk}$.

Using the two claims, we can now prove that the win conditions of both the reduction and the security notion are equivalent, i.e. that for any call to $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}.\mathsf{Verify}(m, \sigma)$, we have that

$$\begin{aligned}& \nu.\mathrm{Verify}(\mathsf{verifier.pk}, (m||\text{“msg”}), \sigma) = \textit{true} \land \\ & (m, \mathsf{verifier.pk}) \notin \mathsf{MsgList} \land \\ & \big(\forall i \in \mathsf{Corrupted}, \exists (j, m_u) \in \mathsf{UpdateList} \text{ s.t.} \\ & \quad j > i \ \land m_u \in \mathsf{RcvdUpdateList}\big) \implies b_{\text{store}} \neq \bot \ .\end{aligned}$$

b_{store} can be set in either (i) the Verify oracle (ii) the $\mathsf{RcvUpdate}$ oracle. We will prove that if the win condition holds and it was not set in (i), it must have been set it (ii) before the call to Verify.

We first establish that if the win condition is true and the current verifier.pk is honest, it is not corrupted, i.e.

$$\begin{aligned}
&\nu.\text{Verify}(\mathsf{verifier.pk}, (m||\text{“msg”}), \sigma) = \mathit{true} \land \\
&(m, \mathsf{verifier.pk}) \notin \mathsf{MsgList} \land \\
&\big(\forall i \in \mathsf{Corrupted}, \exists (j, m_u) \in \mathsf{UpdateList} \text{ s.t.} \\
&\quad j > i \ \land m_u \in \mathsf{RcvdUpdateList}\big) \land \\
&\quad \exists i' \text{ s.t. } pk_{i'} = \mathsf{verifier.pk} \implies i \notin \mathsf{Corrupted}\ .
\end{aligned}$$

This holds because if the win condition is true, there must have been a regular update following the last corruption event. We know from Claim 3 that verifier.pk is honest immediately after the update and any corruption event following the regular update would have invalidated the win condition.

When Verify is called with inputs (m, σ), we can distinguish between two cases. Either (a) verifier.pk is honest or (b) it is not.

If (a) is true and the rest of the win condition holds (especially $(m, \mathsf{verifier.pk}) \notin \mathsf{MsgList}$), then the adversary has forged a signature under an honest key that was not corrupted. b_{store} is then set to b, since as a result of Claim 2, i.e. if a signature was forged under an honest key in $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$.Verify it was also a forgery in $\mathsf{MI\text{-}CEUF\text{-}CMA}^{b,\nu}$.Verify. Thus, in case of (a), the win condition does indeed imply the that $b_{\text{store}} \neq \bot$ and furthermore $b_{\text{store}} = b$.

If (b) is true, then the adversary must have changed verifier.pk such that it is no longer honest, but without corrupting it (which would invalidate the win condition). This means intuitively that the adversary must have forged a valid update message under an honest, and uncorrupted key at some point and thus have broken the signature scheme, leading to b_{store} being set. More formally, let i be the epoch after the most recent regular update. Then Claim 3 shows that we have $\mathsf{verifier.pk} = pk_i$ immediately after the update. Since our premise says that verifier.pk is not true, there must exist an epoch i', where the adversary replaced an honest verifier.pk (with $\mathsf{verifier.pk} = pk_{i'}$) with a public key pk that is not honest. Since changing verifier.pk is only possible through a call to RcvUpdate, the adversary must have called RcvUpdate with input $m_u = (pk, \sigma)$ with a forged update message, i.e. $(pk_{i'}, (pk||\text{“update”})) \notin \mathsf{MsgList}$. If the message had not been forged, the update would have been a regular one and verifier.pk would still be honest. This, together with the fact that verifier.pk is not corrupt and Claim 3 implies that b_{store} is set to b in said call to RcvUpdate, implying that $b_{\text{store}} \neq \bot$, thus proving our claim. □

Having established that $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ provides simulation correctness, only a simple game hop is required to conclude our proof.

Claim 4 (Idealizing $\mathsf{MI\text{-}CEUF\text{-}CMA}^{0,\nu}$)**.** *Let μ be the RSIG construction based on the digital signature scheme ν, then we have for $b \in bin$*

$$\begin{aligned}
&\mathsf{R}_{\mathsf{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CEUF\text{-}CMA}^{0,\nu} \\
&\overset{\epsilon_{\mathsf{MI\text{-}CEUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R}_{\mathsf{PCS\text{-}SIG}})}{\approx} \mathsf{R}_{\mathsf{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CEUF\text{-}CMA}^{1,\nu}.
\end{aligned}$$

Proof. The proof follows from a simple application of Lemma 6 of [14]. More intuitively, it follows directly from our assumption, since $\mathcal{A} \circ \mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ is a valid adversary against MI-CEUF-CMA. □

Finally, Theorem 3 follows from claims 1 and 4. Combining them gives us

$$\begin{aligned}
\mathsf{PCS\text{-}SIG}^{0,\mu} &\equiv \mathsf{R}_{\mathsf{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CEUF\text{-}CMA}^{0,\nu} \\
&\overset{\epsilon_{\mathsf{MI\text{-}CEUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R}_{\mathsf{PCS\text{-}SIG}})}{\approx} \mathsf{R}_{\mathsf{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CEUF\text{-}CMA}^{1,\nu} \\
&\equiv \mathsf{PCS\text{-}SIG}^{1,\mu}.
\end{aligned}$$

□

Remark 1. *The same proof holds if PCS-SIG is strengthened such that* PCS-SIG.Verify *provides SUF-CMA security guarantees as discussed in Section 5.1, and the construction is instantiated with a SUF-CMA secure signature scheme instead of an EUF-CMA secure one. This is because the oracle call to* Verify *in* $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$.RcvUpdate *gives strictly stronger guarantees and the win-condition in* PCS-SIG.Verify *is equivalent, just as in the proof shown above.*

Appendix E. Overview of changes

- **June 2021, v3.0**: Significantly improved version appearing at USENIX 2021:
  1) Title change to better reflect content.
  2) Added section on group messaging security models (MLS + Signal).
  3) Added expected overhead when applying RSIGs in real-world deployments.
  4) Separated design space table for clarity.
  5) Add logical formalism on compromise and healing with explicit freshness condition for use in future security models.
- **Dec 2019, v2.0**: Completely revised and significantly extended version:
  1) Improved problem identification and explanation
  2) Extended exploration of design space
  3) Security definition for PCS-signatures
  4) RSIG construction and proof that it is a PCS-signature
  5) Title change from the old *"Revisiting Post-Compromise Security Guarantees in Group Messaging"* to the new *"Efficient Post-Compromise Security Beyond One Group"*
- **May 2019, v1.0**: Initial release focusing on the discovered discrepancy, without solution details nor full design space considerations.