2019

Efficient Post-Compromise Security Beyond One Group

Cremers, Cas; Hale, Britta; Kohbrok, Konrad

Efficient Post-Compromise Security Beyond One Group

Version 2.0, December 2019*

Cas Cremers
CISPA Helmholtz Center for Information Security
Saarland Informatics Campus
Email: cremers@cispa.saarland

Britta Hale†
Naval Postgraduate School (NPS)
Email: britta.hale@nps.edu

Konrad Kohbrok‡
Aalto University
Email: konrad.kohbrok@aalto.fi

Abstract

Modern secure messaging protocols such as Signal [1] can offer strong security guarantees, in particular Post-Compromise Security (PCS) [2]. The core PCS mechanism in these protocols is designed for pairwise communication, making it inefficient for large groups. To address this, recently proposed designs for secure group messaging, ART [3], IETF's MLS Draft-07 [4]/TreeKEM [5], use group keys derived from tree structures to efficiently achieve PCS in large groups.

In this work we explore the healing behaviors of the pairwise and group-key based approaches. We show that both approaches have inherent limitations to what they can heal, and that without additional mechanisms, both ART and TreeKEM/MLS Draft-07 offer significantly weaker PCS guarantees than those offered by groups based on pairwise PCS channels: for example, we show that if new users can be created dynamically, ART, TreeKEM, and MLS Draft-07 *never* fully heal authentication.

The core underlying problem is that the scope of the healing in ART and MLS is limited to a single group. We lay out the design space of this complex healing problem to identify mechanisms that narrow the gap between the pairwise and group-key approaches, and provide stronger healing for both. Optimizing security and minimizing overhead leads us to a promising solution based on (i) global updates and (ii) *post-compromise secure signatures*.

We provide a security definition for post-compromise secure signatures and an instantiation. Notably, our solution can also be used to improve the healing properties of pairwise protocols such as Signal towards new users who did not previously receive a message of a compromised user.

1. Introduction

Modern secure messaging applications, notably using the Signal protocol libraries [1], can offer very strong security guarantees, including Post Compromise Security (PCS) [2]. Intuitively, PCS describes a protocol's ability to "self-heal" after a participant has been compromised.

Consider the following case: a communicating pair of agents continuously exchange Diffie–Hellman keying material, which is used to update a shared symmetric key, such that the key depends on all the previously exchanged DH material. If an adversary compromises the full state of a participant it can immediately impersonate them, because other participants cannot differentiate between the adversary and the compromised participant. However, if at some later point, the previously compromised participant manages to generate and transfer a new DH share to other participants without the adversary interfering, the next symmetric keys will again be unknown to the adversary, and the session is considered "healed" from that point onwards. In other words, if an adversary wants to eavesdrop or impersonate a compromised participant at some future time, it will have to actively interfere in all of the participant's communications until that time, and continue afterwards to avoid detection. This makes compromising a participant's state a substantially less attractive attack vector, especially for mass surveillance.

Until recently, the protocols that offered PCS were inherently pairwise. To obtain PCS in group messaging, a simple solution is to use the pairwise mechanisms, in which sending a message m in a group of N members is implemented by sending m to each group member individually over $N-1$ pairwise channels. This leads to a complete communication graph and therefore does not scale effectively: m must be encrypted under $N-1$ different keys, scaling the communication and computation overhead for each sender linearly. This also impacts any servers that buffer data until the recipients come online, requiring buffer sizes that 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 do not offer PCS guarantees, and instead fall back to simpler protocols for groups.

The quest for efficient protocols that provide PCS in large groups has driven the design of new protocols. The main two designs are ART [3] and MLS Draft-07 [4], which is based on the TreeKEM protocol [5]. Notably, the ongoing

∗: A summary of changes can be found in Section B

†: *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*

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, which are continuously updated by individual group members to achieve PCS. They offer logarithmic scaling as opposed to linear scaling, making them efficient for large groups.

In this work we revisit the PCS healing properties of both the pairwise and group-key designs in detail. Anticipating some of our results, we provide an illustrative example. Consider a party A whose full state has been compromised, and assume that while the adversary is temporarily passive, A sends an update message in a group G, which is received by all of its members, including B. Afterwards the adversary becomes active again. For both design approaches, this healing update stops the adversary from eavesdropping on G and impersonating A in the group G. However, there are substantial differences, including the following.

In groups based on pairwise Signal, we have that

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

In contrast, in group-key designs such as ART and TreeKEM/MLS Draft-07, we have that

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

As a corollary we have for for the existing group-key designs, that if the adversary can create new identities, regardless of any healing, they can always impersonate A towards any participant $B \in G$, by simply creating a new identity Z and starting a group $\{A, B, Z\}$, and impersonating A towards B in that group. Thus, surprisingly, ART and TreeKEM/MLS currently have no mechanism to fully heal A w.r.t. B, since the adversary can always impersonate A to B in the future.

In this work, we investigate this discrepancy in security guarantees and explore possible solutions. Our main contributions are:

- We uncover a substantial discrepancy 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: for ART and TreeKEM/MLS Draft-07, if participants can be dynamically added to the system, a compromised participant can forever be impersonated towards any other participant, regardless of any healing. This is especially surprising since the group-key designs were explicitly designed to (more efficiently) achieve the same security as the pairwise ones.
- We systematically explore the design space of group messaging protocols with respect to their key update mechanisms and their post-compromise security properties. The design space yields no objectively optimal design, and instead maps out the possibilities and trade-offs available to protocol designers.
- We propose a new primitive, which we coin post-compromise secure signatures, to support the design of protocols with stronger post-compromise security properties, and provide an instantiation. Notably, post-compromise secure signatures can also be used to improve the PCS properties of pairwise protocols.

Outline: In Section 2 we provide an overview of the current group messaging space and ongoing developments, including Signal, MLS, TreeKEM, and ART, before outlining the post-compromise security differences between the main categories of group messaging protocols in Section 3 via 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 gives a definition for ratcheting signatures, defines their security, and provides a construction from SUF-CMA signatures. Finally, we investigate the elevated security achieved using ratcheting signatures within concrete scenarios, and conclude in Section 6.

Related Work: There has been a recent wave of works on post-compromise secure messaging and key exchange, including e.g., [6], [7], [8], but all these works only consider pairwise communication. Some works have explored group messaging in modern protocols [9] but did not consider their PCS properties. ART [3] was the first design aiming to efficiently achieve PCS in group messaging.

Forward secure signature schemes [10] 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 [11], PCS signatures cannot; this serves to illustrate the divide between forward secure and PCS signatures.

While this work introduces the PCS signatures and their use in terms of secure messaging, ideas from forward secure signatures which are relevant for future work include handling untrusted updates [12], and modelling an unpredictable number of time periods [13]. Various other works on forward secure signatures include [14], [15], [16], [17].

2. Background

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

2.1. Post-Compromise Security

The term Post-Compromise Security (PCS) was introduced in [2] 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 [2], [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, where 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). 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.

2.2. Group Messaging in Signal

The core Signal protocol is a pairwise protocol [1], in which two participants continuously send each other asymmetric ratchet updates. In practice, these are ephemeral public keys of the form g^z that get 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, then the adversary is locked out again of subsequent communications since all future keys will depend on that update.

Group messaging in the Signal protocol 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 them in turn.

2.2.1. Groups Using Only Pairwise Channels. Signal groups can be implemented using only pairwise channels. In this case, 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 $A \leftrightarrow D$. Performing an asymmetric update in the group corresponds to sending three individual updates on the respective pairwise channels.

2.2.2. 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 $\mathcal{I}$ generates their own "sender" key $k_{\mathcal{I}}$, which is a symmetric key that is used in a one-to-many fashion within the context of the group. $\mathcal{I}$ then transmits this sender key $k_{\mathcal{I}}$ to the other group members over the pairwise channels. When $\mathcal{I}$ 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 $\mathcal{I}$, 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.

2.3. Group Messaging in ART

ART was the first attempt to provide a group messaging scheme that efficiently provides PCS for large groups [3], [19]. For each group that a party is a member of, a symmetric group key is computed. This key is based on the root of a Diffie-Hellman tree and the previous group key, if it exists.

Key Derivation Using Trees in ART. Each leaf of the tree corresponds to a party, 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). For each member of the group, Alice uses the 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. This enables Alice to compute the entire initial group tree even if the other members are currently offline. The other members can compute their private leaf key by combining their ephemeral private key x with the public setup key g^s. In subsequent updates, members replace those leaves by public keys whose private key only they know.

To achieve PCS, members can update the group key by generating a new key pair for their leaf, and broadcasting the public key to the other members. Additionally, the member computes and sends sufficient update information for the other members to also update their tree. Subsequently, all members compute the new key at the root of the tree, and combine it with the previous group key to obtain the new group key.

Authentication. Authentication is assumed in the ART design, but not concretely specified.

2.4. Group Messaging in TreeKEM/MLS Draft-07

MLS Draft-07 is an ongoing standardization effort by the IETF [4]. Similar to ART, it also employs a tree-based group approach based on TreeKEM [5] for establishing and updating a symmetric group key with the goal of better efficiency than the previously existing approach employed by Signal. In the following section, we will give a simplified overview over the key update mechanism that is used in MLS Draft-07, focusing on the parts relevant for the PCS guarantees achieved by the protocol.

Key Derivation Using Trees in TreeKEM/MLS Draft-07. The tree used in MLS Draft-07 is a left-balanced binary tree. As in ART, every leaf corresponds to one member of the group. Every node is associated with a public key, where the corresponding private key is known to every member in the subtree with that node as root.

To achieve PCS for a particular group, a member can perform an update operation as follows. The 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 key pair for their own leaf node. 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 this process up the tree, deriving new key pairs for every node on the direct path between their leaf node and the root. To distribute the new secret keys of a node to the members in that node's subtree, they use a hybrid public key encryption algorithm to encrypt the secrets used to derive a new key pair to the nearest public key known for every member. Finally, from the path secret in the root node, they derive a new group secret.

Authentication in MLS Draft-07. Every identity is associated with a public signature key and the protocol ensures that members agree on the list of group participants, where each member is represented by their public signature key. Messages involved in performing an update operation as described above are authenticated using this signature key.

3. Security Differences between ART/MLS and Pairwise PCS Channel Groups

In this section we expand on the consequences of the two different design approaches: Approach 1 implementing groups over pairwise channels, and Approach 2 using group-keys as in ART/TreeKEM/MLS Draft-07. We show these differences using concrete scenarios. Henceforth we differentiate between keys used within groups (*local-level* keys) and those used across groups/identity keys (*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 Draft-07 and ART, groups exist independently from one another with the exception of the signature key, which is used in all groups to authenticate each identity.

As this discussion is based on existing protocol designs, we follow the practice of Signal and MLS in assuming long-term signing keys per party and symmetric keys within groups (either pairwise or as a shared group key). Furthermore, we assume that long-term signing 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 in that respect 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.

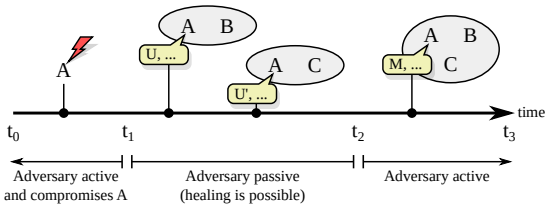
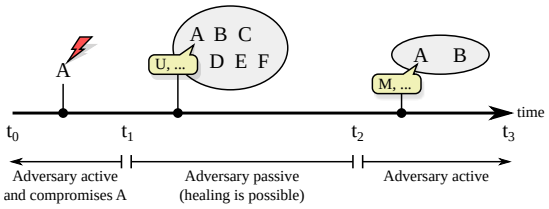
For the purposes of the following discussion in this section, *compromise* refers to full state compromise.

For illustration purposes we consider two scenarios. These are not exhaustive, but suffice to convey the main ideas. The difference between the two approaches with respect to two generic scenarios is summarized in Table 1.

Example 1 (Practical implications)**.** *As a concrete example of the discrepancy in Scenario 2 in Table 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 this heals all future communications with anyone in the company, including all individual channels and subgroups. However, note that 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-07) this 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 (using only pairwise channels), 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 (ART/ MLS Draft-07), Alice is only completely healed once she

Table 1. COMPARING THE POST-COMPROMISE SECURITY PROPERTIES OF GROUPS BUILT FROM PAIRWISE CHANNELS OR ART /MLS STYLE GROUP KEYS

	Scenario 1	**Scenario 2**
		
	A gets compromised before t_1. While the adversary is passive, she sends an update message U to B and an update message 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 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.
	Q: Is M secure?	
Approach 1: All group communication implemented using pairwise channels		
	The messages U and U' "heal" the compromise for the $\{A, B\}$ and $\{A, C\}$ channels; even when adversary becomes active after t_2, they can no longer eavesdrop or insert data on those channels. Furthermore, since the group $\{A, B, C\}$ is built on top of the one-to-one channels, the group has now healed as well. As a result, M is secure. 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.	Sending a message U to the group $\{A, \ldots, F\}$ heals all individual channels among participants and all possible groups consisting of a subset of the original group participants. Because the transmission of M uses the channel that was healed when U was received, M is secure. 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 separate from pairwise channels, e.g., using group keys as in ART, TreeKEM, and MLS-draft-07		
	The messages U and U' heal the one-to-one channels. However, since the group with $\{A, B, C\}$ exists independently of the one-to-one 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 they 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.	The update message 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.

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 they wish to impersonate Alice towards.

These observations point to the fact that PCS per the original definition [2] 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-07, a previously compromised sender's update message only heals the group context it was sent to.

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 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-07, in which groups can be healed by only using updates from the compromised party.

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 approach is to try 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. To improve the situation, we can either (a) try to re-use updates among scopes, or (b) transmit additional update material for different keys and scopes.

We discard option (a) 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 among scopes, instead requiring updates that are additionally linear in the number of groups. Second, from a secure design perspective, and to simplify proofs, we want to maintain maximum key separation.

For our design space in option (b), we consider the following scopes: group scope, and global scope. For the group scope, we have the existing symmetric group keys, and additionally we consider the possibility of the participants generating group-specific asymmetric key pairs. For the global scope, we consider the global asymmetric key pair which is already assumed by existing approaches. We then consider how adding PCS mechanisms to each of these improves the healing behaviour.

4.1. Additional update mechanisms versus update overhead

In Table 2 and Table 3 we summarize the results of the design space. In the columns, we indicate each of the possible key choices for the main PCS update that can occur when A sends a message in group G^i. In the rows, we indicate possible additional key updates that could be jointly triggered with the main update. Note that because we aim for key separation, we assume that these joint updates each have individual update material. The cells subsequently show remaining attack scenarios (i.e. limitations) following the updates and an indication of the overhead of the update mechanism.

Consider Table 2. All updates at the local-level of a given group G^i that do not have any causal effect on updates within other groups result in a *2nd Group Attack*. A 2nd Group Attack example is described in Table 1, in Approach 2 of Scenario 2.

If updates at the local-level of a given group G^i act as a catalyst for updating other groups, the 2nd Group Attack previous described does not hold. However, if such updates are not performed with regularity on a pre-set epoch basis, the result is an information leakage. For example, if A is relatively inactive in G^1 but active in G^2, the members of G^1 will learn information about the activity level of A in G^2. To prevent this, it is possible to enforce update epochs, i.e. updates according to some frequency interval. The epoch may be set at the application level, but must be enforced such that all groups are updated simultaneously. Notably, this may introduce excessive bandwidth and computational overhead, particularly in the case of asymmetric keys, also shown in the table. However, such updates still do not allow PCS healing for future – not yet established – groups.

Table 2. **COMPARISON OF UPDATE SCOPE LIMITATIONS AND OVERHEAD**

Keys updated by causality	**Main update action of A during activity in group G^i**		
	Target group key gk^i	Asymmetric group keys (sk_A^i, pk_A^i)	Global asym. key pair (sk_A, pk_A)
Target group key gk^i	2nd Group Attack $\mathcal{O}(1)$ sym. updates	2nd Group Attack $\mathcal{O}(1)$ asym. updates	
All group keys $\{gk^1, \ldots, gk^N\}$	w/o Epoch: InfoLeak, NF $\mathcal{O}(N)$ sym. updates	w/o Epoch: InfoLeak, NF $\mathcal{O}(N)$ sym. updates	None $\mathcal{O}(N)$ sym. updates
Target group asym. key pair (sk_A^i, pk_A^i)	2nd Group Attack $\mathcal{O}(1)$ asym. updates	2nd Group Attack $\mathcal{O}(1)$ asym. updates	
Group level asym. key pairs $\{(sk_A^1, pk_A^1), \ldots, (sk_A^N, pk_A^N)\}$	w/o Epoch: InfoLeak, NF $\mathcal{O}(N)$ asym. updates	w/o Epoch: InfoLeak, NF $\mathcal{O}(N)$ asym. updates	
Global asym. key pair (sk_A, pk_A)	w/o Epoch: InfoLeak $\mathcal{O}(1)$ asym. updates		None $\mathcal{O}(1)$ asym. updates

Scope limitations and update overhead of various key update combinations. An update in the *Main update action* (column) is combined with an update of the keys in *Keys updated by causality* (row). Internal cells indicate remaining scope limitations, accompanied by associated overhead. A legend is given in Table 3.

Table 3. LEGEND FOR TABLE 2

Notation	Explanation
gk^i	Symmetric key for target group G^i.
(sk^i, pk^i)	Asymmetric key pair for target group G^i.
$\{gk^1, \ldots, gk^N\}$	Group symmetric keys for all groups that A is a member of.
$\{(sk^1, pk^1), \ldots, (sk^N, pk^N)\}$	Group asymmetric keys for all groups that A is a member of.
(sk_A, pk_A)	A's asymmetric global key pair (identity key).
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 $\{G^i, \ldots, G^N\} \setminus \{G^i\}$.
	Non-functionality, e.g. when a global update impacts only a particular group. Additionally, we do not consider cases where both group-level and identity asymmetric keys are used.
w/o Epoch: 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.
NF	No future groups (groups which A has yet to join) achieve PCS following the specified key updates.
None	No scope restrictions, update applies to all groups irrespective of epoch use.
$\mathcal{O}(N)$ 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.

In a similar line of reasoning, we consider updates to group asymmetric keys that cause further updates across all groups. This suffers from the similar information leakage, epoch enforcement and weaknesses for future groups as above, with two added caveats.

1) Using epochs, asymmetric keys would need to be updated per group at regular intervals, resulting in a higher computational overhead than an epoch-based update of symmetric group keys.
2) A record of asymmetric keys would need to be maintained, per group, so that new members can verify the current local-level asymmetric key for each group member.

We also consider updates to local-level keys that act as a catalyst for updating identity keys for A. In this case, there is again potential information leakage about group activity levels to other groups, if update epochs are not used. While this option appears to incur a low computational overhead ($\mathcal{O}(1)$ asymmetric key updates per group key updates), it should be noted that this is per update in G^i. If A wishes to update group keys in all groups $\{G^1, \ldots, G^N\}$, then the identity key must be updated and published after each local-level update in any group.

If updates to the asymmetric identity keys cause updates to per-group symmetric keys, A does not suffer from information leakage or parallel breaks. However, a relatively high overhead is incurred.

The greyed-out cells of Table 2 are out-of-scope cases, such as where updates to identity level keys only affect a particular group. Likewise, we discount cases where both group-level and identity-level asymmetric keys are used.

Finally, we consider updates to the identity's asymmetric key pair, which apply solely to that pair and do not cause

updates to any individual group keys. Notably, this is a single update that applies across all groups simultaneously, thereby reducing computational overhead. Furthermore, such global-level updates apply not only to all current groups but also all future groups, yielding PCS for current groups as well as groups which have not yet been established.

4.2. Update Types

The above discussion points to the efficiency and effectiveness of various update types. It is important to note that both signature key and encryption key updates are necessary for full healing as they achieve different ends. Encryption key updates provide confidentiality PCS while signature key updates provide authenticity PCS. Both goals must be achieved for complete healing. In the context of MLS, for example, confidentiality can be nominally healed with an update; however, if authenticity is not, then an active attacker can still impersonate the compromised party, which in turn is a type of confidentiality break.

Additionally, we note that there are different time scopes with respect to healing in confidentiality/authenticity PCS. Authenticity PCS essentially "locks out" an active attacker. Thus, if authenticity PCS is healed, confidentiality can be healed regardless of whether or not the attacker is active during the confidentiality update phase. Without authenticity PCS, confidentiality PCS can only be achieved under a passive attacker during the update phase – there is no assurance or detection in the case of an active attacker.

4.3. 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 enacting different update type orders following a compromise, before aggregating that view over all possible compromise epochs. For the purposes of this discussion, let $\{G^1, \ldots, G^N\}$ be the set of groups that Alice is a member of.

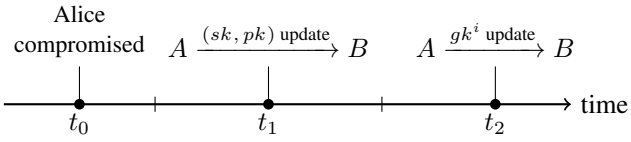


Figure 1. Update order: signature key first.

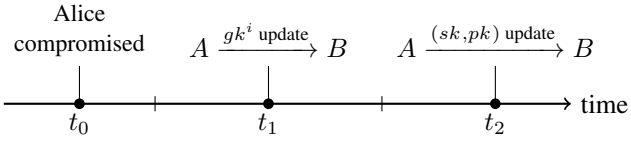


Figure 2. Update order: group key first.

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 G^i. $\mathcal{A}$ can still act as a passive attacker in $\{G^1 \ldots, G^N\} \setminus \{G^i\}$ 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 an a passive attacker in group G^i, but can act as a passive attacker in any group $\{G^1, \ldots, G^N\} \setminus \{G^i\}$ and as an active attacker in G.

$t_2 \rightarrow$ **:** If $\mathcal{A}$ has not injected updates between t_1 and t_2, Alice is healed in G^i. $\mathcal{A}$ can still act as a passive attacker in $\{G^1, \ldots, G^N\} \setminus \{G^i\}$ 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, 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. However, it also affects long-term confidentiality, since an attacker cannot eavesdrop after such encryption updates with ensured validity.

Some applications may prioritize immediate message privacy in a particular group G^i over long-term effects in all groups $\{G^1, \ldots, G^N\}$, or may enforce epoch-based group updates for $\{gk^1, \ldots, gk^N\}$ as discussed in Section 4.1.

Nonetheless, "locking out" an attacker as in Case 1 is arguably preferred, given that its cost/benefit balance is confidentiality in a single group to 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.

4.4. Global-level vs. Local-level Updates

Note that the discussion above points towards a conclusion that differentiates between local-level and global-level updates. 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 [2] 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. 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 can be viewed as aggregated updates to all group keys owned by an identity (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 PCS within a single group without updating global-level keys, as can be seen in MLS Draft-07. However, the PCS guarantees then only apply to the specific group and do not extend to other groups an identity $\mathcal{I}$ may be a member in (i.e. collateral and residual effects as described in Section 3).

Note that, conversely, FS could be considered at the global-level as well, just as PCS may be considered at the local-level only (see Section 3). However, we hold to the claimed security guarantees of existing group messaging protocols for FS demands (see Section 2).

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 signature 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 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 discuss how such signatures can be constructed from standard SUF-CMA signatures. Afterwards, we show that 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 [20]. 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 a strongly 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*.*
- $\mathrm{RSIG.Verify}$ *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* $\mathrm{verify} = 1$ *if* σ_M *is a valid signature of* M *and* $\mathrm{verify} = 0$ *otherwise.*

Correctness. We require that

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

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

PCS-SIG.SignerGen()

assert $sk_1 = \bot$
$\mathsf{sqn} \leftarrow 1$
$sk_{\mathsf{sqn}}, pk_{\mathsf{sqn}} \leftarrow_{\$} \mathrm{RSIG.Gen}$
return pk_{sqn}

PCS-SIG.VerifierGen()

assert $pk_1 \neq \bot$
assert $\mathsf{verifier.pk} = \bot$
$\mathsf{verifier.pk} \leftarrow pk_1$
return $\mathsf{verifier.pk}$

PCS-SIG.Corrupt(i)

assert $sk_i \neq \bot$
$\mathsf{Corrupted} \leftarrow \mathsf{Corrupted} \cup \{i\}$
return sk_i

PCS-SIG.Update()

assert $sk_{\mathsf{sqn}} \neq \bot$
$((sk_{\mathsf{sqn}+1}, pk_{\mathsf{sqn}+1}), m) \leftarrow \mathrm{RSIG.Update}(sk_{\mathsf{sqn}}, pk_{\mathsf{sqn}})$
$\mathsf{sqn} \leftarrow \mathsf{sqn} + 1$
$\mathsf{UpdateList} \leftarrow \mathsf{UpdateList} \cup \{(\mathsf{sqn}, m)\}$
return (pk_{sqn}, m)

PCS-SIG.RcvUpdate(m)

assert $\mathsf{verifier.pk} \neq \bot$
$pk' \leftarrow \mathrm{RSIG.RcvUpdate}(\mathsf{verifier.pk}, m)$
if $pk' \neq \bot$ **then**
  $\mathsf{RcvdUpdateList} \leftarrow \mathsf{RcvdUpdateList} \cup \{m\}$
  $\mathsf{verifier.pk} \leftarrow pk'$
return $\mathsf{verifier.pk}$

PCS-SIG.Sign(m)

assert $sk_{\mathsf{sqn}} \neq \bot$
$\sigma \leftarrow \mathrm{RSIG.Sign}(sk_{\mathsf{sqn}}, m)$
$\mathsf{SigList} \leftarrow \mathsf{SigList} \cup \{(\sigma, m, pk_{\mathsf{sqn}})\}$
return σ

PCS-SIG.Verify(m, σ)

assert $\mathsf{verifier.pk} \neq \bot$
$\mathrm{verify} \leftarrow \mathrm{RSIG.Verify}(\mathsf{verifier.pk}, m, \sigma)$
if $\mathrm{verify} = true \wedge$
  $(\sigma, m, \mathsf{verifier.pk}) \notin \mathsf{SigList} \wedge$
  $\big(\forall sqn' \in \mathsf{Corrupted}$
    $\exists (j, m') \in \mathsf{UpdateList}$ s.t.
    $j > sqn' \wedge$
    $m' \in \mathsf{RcvdUpdateList}\big)$
 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 $\epsilon_{\mathsf{PCS\text{-}SIG}}$-PCS-SIG secure 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, 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 with the public key corresponding to the signer key generated by the SignerGen oracle, the adversary can have the verifier verify signatures using the Verify oracle and update the verifier's public key 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.

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. Instead, we 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 [21], 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 signature was not generated on the message under the recorded public key of the signer using the PCS-SIG.Sign oracle. Note that this requirement fails if the signature, message, or public key do not match (i.e. strong 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 $m \in$ RcvdUpdateList implies that the current verifier key was originally generated by the signer.

Application of PCS-signatures. Although we provide the first formal definition for PCS-signatures, it is conceptionally already used in real world protocols such as DNSSEC [22]. Also, PKI as used in HTTPS aims to provide similar guarantees, albeit with a longer enforced update epoch (i.e. certificates often being set to expire after approximately a year).

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. Note that 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 should be relatively straight-forward to implement such an expiration mechanism by composing PCS-SIG with a secure time-synchronization protocol. Another way of expiring keys is by having the signer publicly revoke keys. 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.

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 [23] to explicitly reveal old authentication keys. For PCS-signatures, explicitly leaking old private signing keys can be used to achieve stronger deniability properties.

5.2. RSIG from a SUF-CMA Secure Signature Scheme

We now introduce an RSIG construction from a SUF-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 by having a party simply sign a given public key, 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 the string "msg" when signing regular messages.

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

$\mu.\mathrm{Gen}(\lambda)$

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

$\mu.\mathrm{Sign}(sk, m)$

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

$\mu.\mathrm{Verify}(pk, m, \sigma)$

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

$\mu.\mathrm{Update}(sk, pk)$

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

$\mu.\mathrm{RecvUpdate}(pk, m)$

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

Figure 4. Description of our construction.

Proof overview. The goal of our proof is to reduce the PCS-SIG-security of the RSIG construction μ to the SUF-CMA-security of the digital signature scheme ν. We begin by "lifting" SUF-CMA twice: First, we introduce corruptible SUF-CMA (CSUF-CMA), which adds a new oracle to SUF-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 CSUF-CMA to its multi-instance version MI-CSUF-CMA, which allows the adversary to interact with $n \in \mathbb{N}$ instances of CSUF-CMA and which was first introduced by Abdalla, Benhamouda and Pointcheval in [17]. For both lifts, we provide straight-forward reductions to SUF-CMA, yielding a combined security loss of factor n. Finally, we build a reduction R by taking the structure of the construction, converting the algorithms to oracles and replacing the calls to the algorithms to the signature scheme ν with calls to MI-CSUF-CMA oracles of the same name. In the same way as PCS-SIG, R also maintains a counter sqn to track which instance of MI-CSUF-CMA it is interacting with. This results in the composition $\mathsf{R} \circ \mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu}$.

This composition $\mathsf{R} \circ \mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu}$ is functionally equal to the $\mathsf{PCS\text{-}SIG}^{b,\mu}$ for $b \in \{0, 1\}$ (i.e. it provides a perfect simulation to the adversary), except that the adversary has an additional way of winning the game when interacting with $\mathsf{R} \circ \mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu}$. This is because the adversary already breaks SUF-CMA security if they manage to forge a valid update message under a key that was previously generated by the Update oracle. We can thus make a game-hop from $\mathsf{PCS\text{-}SIG}$ to $\mathsf{R} \circ \mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu}$, where $b \in \{0, 1\}$ with the adversary gaining at most the advantage equal to that of any adversary breaking SUF-CMA security of ν. Adding the hop from $b = 0$ to $b = 1$ thus yields

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

where $\epsilon_{\mathsf{MI\text{-}CSUF\text{-}CMA}}$ is the advantage function for MI-CSUF-CMA. Summing up the epsilons gives us

$$\epsilon_{\mathsf{PCS\text{-}SIG}}(\mathcal{A}) = 3 \cdot \epsilon_{\mathsf{MI\text{-}CSUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R}_{\mathsf{PCS\text{-}SIG}}),$$

which concludes the proof. We give the full proof in Appendix A.

5.3. Summary of ART and MLS Draft-07 Solutions under RSIG

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-07 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. Additionally, we introduce a new Scenario 3 that showcases how a thus improved ART or MLS Draft-07 provides PCS guarantees beyond what is currently provided.

Let π be the protocol that behaves like MLS Draft-07, using an RSIG scheme instead of a standard signature scheme to authenticate updates. We further require parallelized (i.e. epoch) encryption updates to ensure complete confidentiality

healing in all current groups. 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.

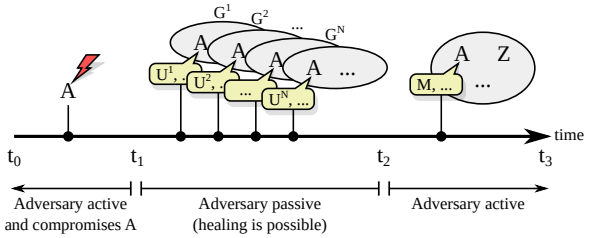


Figure 5. Scenario 3, where $Z \notin \bigcup_{i \in [1,\ldots,N]} G^i$.

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 scenario is depicted in Figure 5.

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

Effects of Approach 2 *in Scenario 3:* Neither MLS Draft-07 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.

6. Conclusions

We have shown 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 meet global-level PCS guarantees. In practice, this means there is a substantial difference between the global-level PCS guarantees offered by groups implemented over pairwise PCS channels and the current proposals for more efficient group key solutions such as ART, TreeKEM, and MLS Draft-07, which only achieve local-level PCS. Notably, our results show that the informally described PCS guarantees in MLS Draft-07 are not met by the current design, leaving it vulnerable to the 2nd Group Attack described in Section 4. Especially since practical use cases for MLS allow the registration of new participants, an adversary that compromised A at some point can always impersonate A in the future towards B, no matter which updates A sends to B.

For MLS, this implies three options as its design stabilises and it moves towards standardization:

1) Explicitly reduce claimed PCS guarantees to a local-level, thereby explicitly allowing attacks on PCS at the global-level.
2) For large groups, explicitly reduce PCS guarantees to a local-level or remove the guarantees entirely, and guarantee global-level PCS for small groups only.
3) Achieve global-level PCS to meet the current goals. This can be done by, for example, (i) concurrent global updates together with (ii) ratcheting signatures, as discussed in Section 4 and Section 5.

Fundamentally, this work demonstrates that it is hard for messaging protocols to achieve PCS in the group setting, and that this stems from the difficulty of achieving authentication PCS as well as a global linkage of updates.

We have show that these weaknesses can be rectified by updating long-term signature keys. We proposed PCS-SIG as a security notion for post-compromise secure signatures and introduced a construction that satisfies it, based on a strongly unforgeable signature scheme.

References

[1] M. Marlinspike and T. Perrin, "The Signal Protocol," Working Draft, Tech. Rep., November 2016. [Online]. Available: https://signal.org/docs/specifications/

[2] K. Cohn-Gordon, C. J. F. Cremers, and L. Garratt, "On Post-compromise Security," in *IEEE 29th Computer Security Foundations Symposium, CSF 2016*, 2016, pp. 164–178.

[3] 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.

[4] R. Barnes, B. Beurdouche, J. Millican, E. Omara, K. Cohn-Gordon, and R. Robert, "The Messaging Layer Security (MLS) Protocol," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-mls-protocol-07, July 2019. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-mls-protocol-07.txt

[5] 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. [Online]. Available: http://prosecco.inria.fr/personal/karthik/pubs/treekem.pdf

[6] 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*, ser. Lecture Notes in Computer Science. Springer, 2019.

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

[8] D. Jost, U. Maurer, and M. Mularczyk, "Efficient ratcheting: Almost-optimal guarantees for secure messaging," 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*, ser. Lecture Notes in Computer Science, Y. Ishai and V. Rijmen, Eds., vol. 11476. Springer, 2019, pp. 159–188.

[9] 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.

[10] M. Bellare and S. K. Miner, "A forward-secure digital signature scheme," in *CRYPTO'99*, ser. LNCS, M. J. Wiener, Ed., vol. 1666. Springer, Heidelberg, Aug. 1999, pp. 431–448.

[11] H. Krawczyk, "Simple forward-secure signatures from any signature scheme," in *ACM CCS 00*, S. Jajodia and P. Samarati, Eds. ACM Press, Nov. 2000, pp. 108–115.

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

[13] T. Malkin, D. Micciancio, and S. K. Miner, "Efficient generic forward-secure signatures with an unbounded number of time periods," in *EUROCRYPT 2002*, ser. LNCS, L. R. Knudsen, Ed., vol. 2332. Springer, Heidelberg, Apr. / May 2002, pp. 400–417.

[14] G. Itkis and L. Reyzin, "Forward-secure signatures with optimal signing and verifying," in *CRYPTO 2001*, ser. LNCS, J. Kilian, Ed., vol. 2139. Springer, Heidelberg, Aug. 2001, pp. 332–354.

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

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

[17] 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.

[18] 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)*, April 2017, pp. 451–466.

[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," in *ACM CCS 18*. ACM Press, 2018, pp. 1802–1819.

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

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

[22] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS security introduction and requirements," Internet Requests for Comments, RFC Editor, RFC 4033, March 2005, http://www.rfc-editor.org/rfc/rfc4033.txt.

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

Appendix A. 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 [21], which we will briefly state here.

We bundle oracles into *packages*, where a pacakge M provides access to a set of oracles $\{\mathsf{O}_1, \mathsf{O}_2, \dots\}$ that operate on a shared package state. This state can only be accessed by oracles of that package. Packages can be composed, such that oracles provided by a package M can call oracles provided by another package M'. For example, let M be

the package representing an assumption and R the package representing a reduction that interacts with the assumption, while at the same time providing oracles for an adversary to call. Then $\mathsf{R} \circ \mathsf{M}$ would represent the composed reduction that an adversary can interact with. We denote with $\epsilon_{\mathsf{M}}(\mathcal{A})$ the function that returns the advantage of an adversary $\mathcal{A}$ in distinguishing packages M^0 and M^1. Following the concrete security paradigm, our goal is to relate the advantage function for our top-level security notion parameterized by our construction to that of the underlying assumption used by our construction. For example, let M' be the top-level security notion and M the assumption our construction is based on. The our goal would be to find a reduction R such that for all adversaries $\mathcal{A}$ against M', we have that $\epsilon_{\mathsf{M}'}(\mathcal{A}) \leq \epsilon_{\mathsf{M}}(\mathcal{A} \circ \mathsf{R})$. Intuitively, it says that we can compose every adversary $\mathcal{A}$ against M' with R such that it becomes an adversary against M.

Continuing the example, the typical proof structure of a proof written in the SSP framework would consist of two steps:

1) Constructing a reduction R such that for $b \in \{0,1\}$, we have $\mathsf{M}'^b \equiv \mathsf{R} \circ \mathsf{M}^b$, where by $\equiv$, we denote perfect equivalence. This step is essentially proving "simulation correctness" of our reduction.
2) Idealizing our assumption(s), i.e. using Lemma 28 of [21] to prove that $\mathsf{R} \circ \mathsf{M}^0 \overset{\epsilon_{\mathsf{M}}}{\approx} \mathsf{R} \circ \mathsf{M}^1$.

In our proof, we will deviate slightly from this structure, by providing a reduction that leaves a gap between the top-level security notion and the composed reduction which is equivalent to the distinguishing gap of our assumption.

A.1. Assumptions

Before we state our main security theorem, we start by stating our standard assumption. We then reduce corruptible SUF-CMA (CSUF-CMA) to standard SUF-CMA and lift CSUF-CMA to a multi-instance version of itself (MI-CSUF-CMA) using a standard hybrid argument. MI-CSUF-CMA is equivalent to the security notion M-SUF-CMA, first introduced by Abdalla, Benhamouda and Pointcheval in [17]. Note, that although functionally equivalent, our version is using a slightly different style to fit the SSP methodology. Finally, we reduce PCS-SIG to MI-CSUF-CMA.

Definition 3 (Standard SUF-CMA Assumption)**.** *Let ν be a digital signature scheme. Then we say ν is $\epsilon_{\mathsf{SUF\text{-}CMA}}$-SUF-CMA secure if for all adversaries $\mathcal{A}$ we have that*

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

where for $b \in \{0,1\}$ the oracles provided by $\mathsf{SUF\text{-}CMA}^{b,\nu}$ are defined as follows:

$\mathsf{Gen}()$
$pk, sk \leftarrow_{\$} \nu.\mathrm{Gen}()$
return pk

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

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

We proceed with the definition of corruptible SUF-CMA, where the only difference between the traditional SUF-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 4 (Corruptible SUF-CMA)**.** *Let ν be a digital signature scheme. Then we say ν is $\epsilon_{\mathsf{CSUF\text{-}CMA}}$-CSUF-CMA secure if for all adversaries $\mathcal{A}$ we have that*

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

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

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

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

Proof. Let $\mathsf{R}_{\mathsf{CSUF\text{-}CMA}}$ be a package that forwards all queries to the SUF-CMA game and returns a fixed random value upon any query to the Corrupt oracle. Let $\mathcal{A}$ be an adversary against CSUF-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

Gen()	Sign(m)	Verify(m, σ)
assert $sk = \bot$	**assert** $sk \neq \bot$	**assert** $pk \neq \bot$
$pk, sk \leftarrow_\$ \nu.\text{Gen}()$	$\sigma \leftarrow \nu.\text{Sign}(sk, m)$	$\text{verify} \leftarrow \nu.\text{Verify}(pk, m, \sigma)$
return pk	$\mathsf{SigList} \leftarrow \mathsf{SigList} \cup (\sigma, m)$	**if** $(\sigma, m) \notin \mathsf{SigList} \wedge \text{verify} = \mathit{true} \wedge \mathsf{C} \neq 1$ **then**
	return σ	**return** b
Corrupt()		**return** v
assert $sk \neq \bot$		
$\mathsf{C} \leftarrow 1$		
return sk		

Figure 6. Description of CSUF-CMA oracles.

received b before that point in time, or the best they can do is guess b at random regardless of the return value of any queries to Corrupt. Thus, the advantage of $\mathcal{A}$ against CSUF-CMA is upper bounded by its advantage against $\mathsf{R}_{\mathsf{CSUF\text{-}CMA}} \circ \mathsf{SUF\text{-}CMA}$. □

Definition 5 (Multi-Instance Corruptible SUF-CMA)**.** *Let ν be a digital signature scheme. Then we say ν is $\epsilon_{\mathsf{MI\text{-}CSUF\text{-}CMA}}$-MI-CSUF-CMA secure if for all adversaries $\mathcal{A}$ we have that*

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

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

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

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

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

Proof. The proof is a simple instantiation of Lemma 28 in [21], essentially a hybrid argument over n $\mathsf{CSUF\text{-}CMA}^{b,\nu}$ packages. The hybrid argument gives us a reduction $\mathsf{R}_{\mathsf{MI\text{-}CSUF\text{-}CMA}}$ such that $\forall \mathcal{A}, \epsilon_{\mathsf{MI\text{-}CSUF\text{-}CMA}}(\mathcal{A}) \leq n \cdot \epsilon_{\mathsf{CSUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R}_{\mathsf{MI\text{-}CSUF\text{-}CMA}})$. Using Theorem 1 concludes the proof. □

Note, that Lemma 28 of [21] defines n as static, i.e. the adversary can not decide how many instances to interact with dynamically. Indstead, 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.

A.2. Main Theorem and Proof

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

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

$$\epsilon_{\mathsf{PCS\text{-}SIG}}(\mathcal{A}) \leq 3n \cdot \epsilon_{\mathsf{MI\text{-}CSUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R}_{\mathsf{PCS\text{-}SIG}}),$$

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

Proof. Let μ be the RSIG construction based on the digital signature scheme ν. We will first define a reduction $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ as shown in Figure 7 that interfaces with $\mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu}$ and that presents the same interface to the adversrary as PCS-SIG. Since we have that $\mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu} = \prod_{i=1}^{n} \mathsf{CSUF\text{-}CMA}_i^{b,\nu}$, in $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ we will call the oracles of the individual $\mathsf{CSUF\text{-}CMA}^{b,\nu}$ packages. We will then inline the definition of the construction into PCS-SIG and the definition of CSUF-CMA into $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ to prove that

$$\mathsf{R}_{\mathsf{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu} \overset{\epsilon_{\mathsf{MI\text{-}CSUF\text{-}CMA}}(\mathcal{A})}{\approx} \mathsf{PCS\text{-}SIG}^{b,\mu}.$$

$\mathsf{R}_{\text{PCS-SIG}}.\mathsf{SignerGen}()$

$pk_1 \leftarrow \mathsf{Gen}_1()$
$\mathsf{sqn} \leftarrow 1$
return pk_{sqn}

$\mathsf{R}_{\text{PCS-SIG}}.\mathsf{VerifierGen}()$

assert $pk_1 \neq \perp$
assert $\mathsf{verifier.pk} = \perp$
$\mathsf{verifier.pk} \leftarrow pk_1$
return verifier.pk

$\mathsf{R}_{\text{PCS-SIG}}.\mathsf{Corrupt}(pk)$

assert $\exists i$ s.t. $pk_i = pk$
$sk \leftarrow \mathsf{Corrupt}_i()$
return sk

$\mathsf{R}_{\text{PCS-SIG}}.\mathsf{Update}()$

$pk_{\mathsf{sqn}+1} \leftarrow \mathsf{Gen}_{\mathsf{sqn}+1}()$
$\sigma \leftarrow \mathsf{Sign}_{\mathsf{sqn}}(pk_{\mathsf{sqn}+1}||\text{“update”})$
$m \leftarrow (pk_{\mathsf{sqn}+1}, \sigma)$
$\mathsf{sqn} \leftarrow \mathsf{sqn} + 1$
return (pk_{sqn}, m)

$\mathsf{R}_{\text{PCS-SIG}}.\mathsf{Sign}(m)$

$\sigma \leftarrow \mathsf{Sign}_{\mathsf{sqn}}(m||\text{“msg”})$
return σ

$\mathsf{R}_{\text{PCS-SIG}}.\mathsf{Rcvupdate}(m)$

assert $\mathsf{verifier.pk} \neq \perp$
$(pk', \sigma) \leftarrow m$
if $\exists i \in \mathbb{N}$ s.t. $pk_i = \mathsf{verifier.pk}$ **then**
  $\text{verify} \leftarrow \mathsf{Verify}_i((pk'||\text{“update”}), \sigma)$
else
  $\text{verify} \leftarrow \nu.\text{Verify}(\mathsf{verifier.pk}, (pk'||\text{“update”}), \sigma)$
if $\text{verify} = true$ **then**
  $\mathsf{verifier.pk} \leftarrow pk'$
return v

$\mathsf{R}_{\text{PCS-SIG}}.\mathsf{Verify}(m, \sigma)$

assert $\mathsf{verifier.pk} \neq \perp$
if $\exists i \in \mathbb{N}$ s.t. $pk_i = \mathsf{verifier.pk}$ **then**
  $\text{verify} \leftarrow \mathsf{Verify}_i((m||\text{“msg”}), \sigma)$
else
  $\text{verify} \leftarrow \nu.\text{Verify}(\mathsf{verifier.pk}, (m||\text{“msg”}), \sigma)$
return v

Figure 7. Description of the oracles of $\mathsf{R}_{\text{PCS-SIG}}$.

In more traditional proofs, this corresponds to showing that our reduction simulates the game correctly for the adversary. Since the simulation is not perfect, we will instead prove, that reduction and security notion are distinguishable only with advantage $\epsilon_{\text{MI-CSUF-CMA}}$. Once we have established this, we can use Lemma 6 of [21] to prove the theorem.

We proceed by inlining the construction and the individual oracle code of the oracles provided by $\mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu}$. See Figure 8 for a side-by-side comparison of the oracles of $\mathsf{PCS\text{-}SIG}^{b,\mu}$ and $\mathsf{R}_{\text{PCS-SIG}}$, where we have inlined the oracles of $\mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu}$. We have renamed annotated the package-internal variables of the $\mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu}$ with their index i as subscript.

For convenience, we will call PCS-SIG the “left side” and $\mathsf{R}_{\text{PCS-SIG}}$ the “right side”. We begin by comparing both sides with the aim of establishing that an adversary interacting with the left side has an equal or lower probability of winning than an adversary interacting with the right side. We begin with the “signer” oracles, i.e. with $\mathrm{Update}, \mathrm{Sign}$ and $\mathrm{Corrupt}$. First, note that the sequence numbers that enumerate the keys of the signer on the left side correspond to the $\mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu}$ instances on the right side. In Figure 8 we have inlined the oracle code into the reduction, showing that the Update oracle is functionally equivalent on both sides except for the assignment of some state variables on the left side. Note, that the corruption state is the same on both sides as well.

Whenever a key at sequence number i is compromised on the left side, the same key is marked as compromised in instance i on the right side. Similarly, whenever a key is used to sign a message on the left side, the same key is used to sign the message on the right side, leading to consistent SigList states on both sides. Note that this is only true for messages tagged “msg”. For convenience, we will call keys that have been generated by the Update oracle “honest” and all other keys “dishonest”. As a corollary, note that all honest keys correspond to an $\mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu}$ instance on the right side.

Comparing the behaviour of the verifier (i.e. the $\mathrm{RecvUpdate}$ and Verify oracles) on both sides is a little less straight forward. However, note, that both sides are functionally the same, with differences only in the conditions under which the adversary wins.

Looking at the $\mathrm{RecvUpdate}$ oracle on the left side, we can see that the adversary can make the verifier change its current public key verifier.pk by submitting a public key and a valid signature of the key concatenated with the “$update$” string. Note, that even if the current verifier.pk is honest, the adversary does not immediately win if they manage to forge a valid signature. The $\mathrm{RecvUpdate}$ on the right side is functionally identical, except that additionally, the adversary wins if the current verifier.pk is honest ($\exists i \in \mathbb{N}$ s.t. $pk_i = \mathsf{verifier.pk}$), they submit a signature that is valid ($v = 1$) and is forged ($(\sigma, (pk'||\text{“update”})) \notin \mathsf{SigList}_i$). We will see that this difference counteracts another difference in the Verify oracle.

Looking at the Verify oracle, it is again functionally the same on both sides except for the win condition. On the left side, it states that the adversary wins if the adversary submits a valid forgery (i.e. not created previously by the Sign oracle) and after every corruption, there has been at least one “regular” update, where by regular we denote the event that an update message is generated by the Update oracle and received by the $\mathrm{RecvUpdate}$ with a successful signature validation. On the right side, the current verifier key has to be honest, not corrupted and the signature submitted to the oracle must be a valid forgery. To prove that these win conditions are equivalent (taking into account the previously noted difference in the $\mathrm{RecvUpdate}$ oracle,) it remains to show that (i) after every corruption a successful update has

PCS-SIG.Sign(m)

assert $sk_{\mathsf{sqn}} \neq \bot$
$\sigma \leftarrow \nu.\mathrm{Sign}(sk_{\mathsf{sqn}}, (m||\text{“msg”}))$
$\mathsf{SigList} \leftarrow \mathsf{SigList} \cup \{(\sigma, m, pk_{\mathsf{sqn}})\}$
return σ

PCS-SIG.Corrupt(i)

assert $sk_i \neq \bot$
$\mathsf{Corrupted} \leftarrow \mathsf{Corrupted} \cup \{i\}$
return sk_i

PCS-SIG.Update()

assert $sk_{\mathsf{sqn}} \neq \bot$
$(pk_{\mathsf{sqn}+1}, sk_{\mathsf{sqn}+1}) \leftarrow \nu.\mathrm{Gen}(\lambda)$
$\sigma \leftarrow \nu.\mathrm{Sign}(sk_{\mathsf{sqn}}, (pk_{\mathsf{sqn}+1}||\text{“update”}))$

$m \leftarrow (pk_{\mathsf{sqn}+1}, \sigma)$
$\mathsf{sqn} \leftarrow \mathsf{sqn} + 1$
$\mathsf{UpdateList} \leftarrow \mathsf{UpdateList} \cup \{(\mathsf{sqn}, m)\}$
return (pk_{sqn}, m)

PCS-SIG.RcvUpdate(m)

assert verifier.pk $\neq \bot$
$(pk', \sigma) \leftarrow m$

$\mathrm{verify} \leftarrow \nu.\mathrm{Verify}(\mathsf{verifier.pk}, (pk'||\text{“update”}), \sigma)$
if $\mathrm{verify} = true$ **then**
  $\mathsf{RcvdUpdateList} \leftarrow \mathsf{RcvdUpdateList} \cup \{m\}$
  $\mathsf{verifier.pk} \leftarrow pk'$
return verifier.pk

PCS-SIG.Verify(m, σ)

assert verifier.pk $\neq \bot$
$\mathrm{verify} \leftarrow \nu.\mathrm{Verify}(\mathsf{verifier.pk}, (m||\text{“msg”}), \sigma)$
if $\mathrm{verify} = true \wedge$
  $(\sigma, m, \mathsf{verifier.pk}) \notin \mathsf{SigList} \wedge$
  $\big(\forall sqn' \in \mathsf{Corrupted}$
    $\exists (j, m') \in \mathsf{UpdateList}$ s.t.
    $j > sqn' \wedge$
    $m' \in \mathsf{RcvdUpdateList}\big)$
 then
  return b
return verify

$\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$.Sign($m$)

assert $sk_{\mathsf{sqn}} \neq \bot$
$\sigma \leftarrow \nu.\mathrm{Sign}(sk_{\mathsf{sqn}}, m||\text{“msg”})$
$\mathsf{SigList}_{\mathsf{sqn}} \leftarrow \mathsf{SigList}_{\mathsf{sqn}} \cup (\sigma, (m||\text{“msg”}))$
return σ

$\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$.Corrupt($i$)

assert $sk_i \neq \bot$
$\mathsf{C}_i \leftarrow 1$
return sk_i

$\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$.Update()

assert $sk_{\mathsf{sqn}} \neq \bot$
$pk_{\mathsf{sqn}+1}, sk_{\mathsf{sqn}+1} \leftarrow_{\$} \nu.\mathrm{Gen}()$
$\sigma \leftarrow \nu.\mathrm{Sign}(sk_{\mathsf{sqn}}, (pk_{\mathsf{sqn}+1}||\text{“update”}))$
$\mathsf{SigList}_{\mathsf{sqn}} \leftarrow \mathsf{SigList}_{\mathsf{sqn}} \cup (\sigma, (pk_{\mathsf{sqn}+1}||\text{“update”}))$
$m \leftarrow (pk_{\mathsf{sqn}+1}, \sigma)$
$\mathsf{sqn} \leftarrow \mathsf{sqn} + 1$

return (pk_{sqn}, m)

$\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$.RcvUpdate($m$)

assert verifier.pk $\neq \bot$
$(pk', \sigma) \leftarrow m$
if $\exists i \in \mathbb{N}$ s.t. $pk_i = \mathsf{verifier.pk}$ **then**
  $\mathrm{verify} \leftarrow \nu.\mathrm{Verify}(pk_i, (pk'||\text{“update”}), \sigma)$
  if $(\sigma, (pk'||\text{“update”})) \notin \mathsf{SigList}_i \wedge \mathrm{verify} = true \wedge \mathsf{C}_i \neq 1$ **then**
    return b
else
  $\mathrm{verify} \leftarrow \nu.\mathrm{Verify}(\mathsf{verifier.pk}, (pk'||\text{“update”}), \sigma)$
if $\mathrm{verify} = true$ **then**

  $\mathsf{verifier.pk} \leftarrow pk'$
return verifier.pk

$\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$.Verify($m, \sigma$)

assert verifier.pk $\neq \bot$

if $\exists i \in \mathbb{N}$ s.t. $pk_i = \mathsf{verifier.pk}$ **then**
  $\mathrm{verify} \leftarrow \nu.\mathrm{Verify}(pk_i, (m||\text{“msg”}), \sigma)$
  if $(\sigma, m) \notin \mathsf{SigList}_i \wedge \mathrm{verify} = true \wedge \mathsf{C}_i \neq 1$ **then**
    return b
else
  $\mathrm{verify} \leftarrow \nu.\mathrm{Verify}(\mathsf{verifier.pk}, (m||\text{“msg”}), \sigma)$
return verify

Figure 8. Comparison between PCS-SIG with inlined construction and $\mathsf{R}_{\mathsf{PCS\text{-}SIG}}$ with inlined MI-CSUF-CMA oracles.

happened if and only if (ii) the current verifier key is honest and not corrupted.

We prove this in two steps. First, we show that (i) $\implies$ (ii), followed by a proof that (ii) $\implies$ (i).

Examining the $\mathrm{RecvUpdate}$ oracle on the left side, we can see that a regular update means that an honest public

key has been installed at the verifier. For (i) to be true, the current verifier key thus has to be honest and uncorrupted. After that, whenever the adversary manages to make the verifier accept a new public key with a signature valid under an honest key, they have either (a) performed another regular update or (b) forged a valid signature for their own (dishonest) key that is valid under the current, honest verifier key. In the case of (a), the new verifier key is again honest. In case of (b), the adversary triggers the win condition on the right side. Thus (i) $\implies$ (ii) either holds or we have a successful adversary against $\mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu}$ and would be able to distinguish between $\mathsf{R_{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CSUF\text{-}CMA}^{0,\nu}$ and $\mathsf{R_{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CSUF\text{-}CMA}^{1,\nu}$.

For (ii) $\implies$ (i), recall from the previous paragraph that the verifier key is honest if and only if a regular update has taken place. Since (ii) states that the current verifier key is not corrupted, there must have been at least one successful update after the most recent corruption thus proving (ii) $\implies$ (i).

This means that for all adversaries $\mathcal{A}$ the distinguishing advantage between $\mathsf{R_{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu}$ and $\mathsf{PCS\text{-}SIG}^{b,\mu}$ is $\epsilon_{\mathsf{MI\text{-}CSUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R_{PCS\text{-}SIG}})$, i.e.

$$\forall b \in \{0,1\} :$$
$$\mathsf{R_{PCS\text{-}SIG}} \circ \mathsf{MI\text{-}CSUF\text{-}CMA}^{b,\nu} \overset{\epsilon_{\mathsf{MI\text{-}CSUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R_{PCS\text{-}SIG}})}{\approx} \mathsf{PCS\text{-}SIG}^{b,\mu}.$$

Since Lemma 6 of [21] gives us

$$\mathsf{MI\text{-}CSUF\text{-}CMA}^{0,\nu} \overset{\epsilon_{\mathsf{MI\text{-}CSUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R_{PCS\text{-}SIG}})}{\approx} \mathsf{MI\text{-}CSUF\text{-}CMA}^{1,\nu}$$

We thus have for all adversaries $\mathcal{A}$ against $\mathsf{PCS\text{-}SIG}^{b,\mu}$

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

Summing up the distinguishing advantages gives us

$$\begin{aligned}&\epsilon_{\mathsf{PCS\text{-}SIG}}(\mathcal{A}) \\ &= 3 \cdot \epsilon_{\mathsf{MI\text{-}CSUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R_{PCS\text{-}SIG}}) \\ &\leq 3n \cdot \epsilon_{\mathsf{SUF\text{-}CMA}}(\mathcal{A} \circ \mathsf{R_{PCS\text{-}SIG}} \circ \mathsf{R_{MI\text{-}CSUF\text{-}CMA}} \circ \mathsf{R_{CSUF\text{-}CMA}}).\end{aligned}$$

Note, that since we reduce to MI-CSUF-CMA, we are restricted to the class of adversaries that does n queries to the Update oracle. However, as noted earlier we can instantiate the reduction any $n \in \mathbb{N}$, the theorem indeed holds for all adversaries. □

Appendix B.
Overview of changes

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.