# How fast do you heal?
# A taxonomy for post-compromise security in secure-channel establishment

Olivier Blazy
*LIX, CNRS, Inria, École Polytechnique, Institut Polytechnique de Paris, France*

Ioana Boureanu
*Univ. of Surrey, Surrey Centre for Cyber Security, UK*

Pascal Lafourcade
*University of Clermont Auvergne, France*

Cristina Onete
*University of Limoges, France*

Léo Robert
*University of Clermont Auvergne, France*

## Abstract

Post-Compromise Security (PCS) is a property of secure-channel establishment schemes, which limits the security breach of an adversary that has compromised one of the end-point to a certain number of messages, after which the channel *heals*. An attractive property, especially in view of Snowden's revelation of mass-surveillance, PCS was pioneered by the Signal messaging protocol, and is present in OTR. In this paper, we introduce a framework for *quantifying and comparing* PCS security, with respect to a broad taxonomy of adversaries. The generality and flexibility of our approach allows us to model the healing speed of a broad class of protocols, including Signal, but also an identity-based messaging protocol named SAID, and even a composition of 5G handover protocols.

## 1 Introduction

Secure-channel establishment is a cornerstone of modern communication. It allows users to exchange messages whose confidentiality and authenticity are guaranteed, even with respect to potential Person-in-the-Middle attackers. Authenticated Key-Exchange (AKE) protocols have, for decades now, enabled such channels to be established: for Internet browsing (TLS protocol), for mobile networks (AKA protocol), for secure remote access to another machine (SSH), and so on.

The revelations of Edward Snowden, who exposed the reality of mass surveillance by security agencies such as the NSA or GCHQ, were a boost to the widespread deployment of secure channels. It is now confirmed that powerful adversaries can, and do, fully corrupt the private information stored on a targeted device, thus learning most (if not all) of its secrets. Even then, secure channels open *prior* to the adversary's intrusion can still preserve confidentiality and authenticity if Perfect Forward Secrecy (PFS) is guaranteed. Unfortunately, however, all sessions *following* the party's compromise will no longer guarantee either confidentiality or authenticity.

The lack of future security is particularly problematic for secure channels that are meant to last for a long time, such as those generated by asynchronous messaging applications. Say that a civilian, Alice, has a journalist friend, Bob, with whom she communicates via a secure messaging application. While abroad, Alice receives sensitive documents from a whistle-blower, whose request is that she sends them to Bob. She messages Bob about them. But as she travels back home, Alice's phone might be compromised at border control. At this point, she would like to have three guarantees: that her past communication with Bob is secure; that no one can impersonate her to bait Bob; and that in a little while, she will be able to resume talking to Bob without (for instance) destroying her phone.

The PFS of the channel could guarantee the first of these requirements. For the second and third, Alice requires a property pioneered by Marlinspike and Perrin in the context of the Signal protocol [22], called *Post-Compromise Security* (PCS) by Cohn-Gordon *et al.* [15]. This attractive feature implies that the secure channel established in Signal by Alice and Bob can repair (or "*heal*") its security, even after a full compromise.

But of course, Alice wants to know: how fast will the channel's security return, and under what conditions?

Cohn-Gordon *et al.* [13] showed that in the original Signal protocol, Alice can recover security after she and Bob have switched speakers (*i.e.*, exchanged sender/receiver roles) twice in the conversation. So, if Bob was the current sender when Alice's phone was compromised, then Alice must first send (at least) a message, and wait for Bob's reply before they are safe again. All messages sent between the moment of corruption and Bob's second reply are compromised by the attacker.

Even more problematic is the case in which the attacker uses the data recovered from Alice to insert itself into the communication, choosing whether it wants to just impersonate Alice to Bob, impersonate Bob to Alice, or both, and set itself up as a permanent Person-in-the-Middle between them. For active attackers, Alice's conversation with Bob will *never* heal.

Hailed as a revolutionary design in secure-channel establishment, Signal was used as a basis for group messaging schemes, such as ART [14] and MLS [5]; the protocol can also be used directly if group messaging is implemented as a composition of pairwise secure channel between all the participants. Other messaging protocols, such as OTR [9], Matrix [1], Wire [21], also guarantee some measures of PCS.

To improve the healing speed of Signal, Blazy *et al.* [7] used identity-based cryptography and introduced SAID, a protocol which provides much better security against active attacks. As SAID relies on a different paradigm than Signal, it is hard to compare their respective security levels. Yet, at a high level both protocols guarantee post-compromise security – so are they perhaps equivalent?

In our chosen example, the healing speed makes a huge difference to Alice. She would moreover ideally like to know that healing depends entirely on her (rather than, say, on Bob returning online). Finally, from a designer's point of view it is crucial to understand how different protocols handle different adversaries: would the attacker only require short-term (potentially more vulnerable) values, or does it need long-term secrets?

**Our contributions.** We propose a *metric* allowing to assess and compare the post-compromise security of apparently-incomparable protocols, such as Signal and SAID. Our aim is to quantify the healing speed and healing conditions of a scheme with respect to various *classes* of attackers.

More precisely, our contribution is threefold: we formally define a broad category of two-party protocols that enable key-evolution (which we call SCEKE, for Secure-Channel Establishment schemes with Key-Evolution); we then define a framework, consisting of a *post-compromise security (PCS) metric* and several classes of adversaries, varying in strength and abilities; and we provide a comparison of three SCEKE protocols, thus exemplifying our framework.

For this, we introduce a taxonomy of adversaries, in terms of 3 characteristics: their *access* (is it a trusted party or not), their *power* (active or passive), and their *reach* (which values does it compromise?). A weak adversary may only be able to compromise stage-specific values (which are always in memory). A very strong adversary might be a trusted party (like Signal's credential server), able to actively hijack sessions and fully compromise all the data belonging to a party.

Our PCS adversaries attack SCEKE protocols, in which channel keys are only used for a short time (*i.e.*, during a *stage*). We index stages by pairs of positive integers $(x, y)$, and consider an evolution of stages that is either *horizontal* (from stage $(x, y)$ to $(x+1, y)$), or *vertical* (from stage $(x, y)$ to $(1, y+1)$). Stages with the same $y$ value are said to be on the same chain.

PCS adversaries will target a specific channel (also called message) key, having compromised an endpoint to that channel. Our metric measures the *number of messages* required, per message-chain, for the security of the channel to heal after this corruption. Thus, a protocol is $(\infty, 1)$-PCS resistant with respect to some class of adversaries if the honest parties lose channel security for all the stages obtained through horizontal evolution, and at most 1 stage obtained through vertical evolution, starting from the last stage $(x^*, y^*)$ at which the adversary compromised either endpoint. Optimal healing corresponds to $(1, 0)$-PCS security, while the worst healing is $(\infty, \infty)$-PCS security, *i.e.*, the protocol's security never heals.

To showcase the broad reach of our metric, we use it to compare 3 schemes that would otherwise be hard to compare: the PKI-based Signal asynchronous messaging protocol (analyzed by Cohn-Gordon *et al.* [13]), the identity-based SAID asynchronous-messaging protocol [7], and the 5G handover protocols in mobile networks [2, 3]. For the latter protocol, we are the first (to our knowledge) to model and analyze the post-compromise security afforded by sequential compositions of handovers. We also show how to easily tweak 5G handovers to obtain much faster healing. Our results are summarized in Fig. 4. Our framework can also be used to analyse the security of many other PCS protocols.

**Related work.** Provable-security analyses of known protocols, such as those of Signal [13] and SAID [7] are cornerstones to our work, and enable us to show how our framework compares with existing results. This is one of the reasons those two protocols were chosen. However, in our work we go beyond current results, both in terms of scope and of provable results.

Although comparatively infrequent, taxonomies and metrics exist and are very useful in cryptography. An eminent example is the taxonomy of privacy notions by Pfitzmann and Hansen [23], which ranks and classifies subtly-different terms referring to user privacy (such as anonymity, privacy, unlinkability, undetectability, etc.). We choose to focus on the property of channel security in the context of a particular type of scheme; moreover, while we classify attacks by three *types* of parameters, our taxonomy focuses on a precise *quantification* of healing speed, which is out of scope for [23].

Our methodology better resembles taxonomy efforts such as [17, 20], whose purpose is to categorize security definitions in information-flow, or electronic voting, respectively. Our work, however, focuses on a very different type of protocol than in these two fields; moreover, we use a provable-security, rather than a formal-methods approach.

The mechanics of our model resemble those of Fischlin and Günther [19], which extend prior work [6, 12] to complex, multi-stage key-agreement. Our main security notion (Post-Compromise Security), however, is atypical for [19]; moreover, we are the first to define the generic notion of SCEKE protocols. A parallel line of work recently introduced by Brzuska *et al.* [10] analyzes, in a compositional framework (using state-separating proofs [11]), the security of the multi-party asynchronous messaging protocol MLS. Unlike that technique, ours is not composable[1], but unlike [10] we do take into account authentication, which is crucial in the case of active adversaries.

Finally, we note that our main contribution in this work is the taxonomy of attackers and quantification of PCS-security. This is why we carefully chose only three protocols (with apparently incomparable degrees of PCS) to analyze: Signal,

[1]This is, technically speaking, because we quantify PCS-security within the *winning conditions*.

SAID, and the 5G Handover protocols. We discuss how our framework applies to other protocols below.

**Other protocols.** Although we choose to showcase our metric by means of the three protocols cited above, our framework can be applied to other PCS-providing protocols, such as OTR, Matrix' Olm protocol for 2-party rooms, and Wire. Signal ratchets are actually a combination of OTR and SCIMP ratchets[2]. Notably, the former provides PCS security. However, note that OTR's focus is privacy, not necessarily (PCS-)security, and thus limits and encrypts any explicit long-term-authentication steps. This gives it a relatively weak security in our framework when we consider active adversaries, but interestingly provides less advantages for insider attackers. An interesting future research question is how to optimally balance the kind of privacy desired by OTR and its PCS healing speed.

The Olm protocol used by Matrix resembles Signal (some differences exist with respect to the type of keys used, signed or unsigned) and would provide similar metric results in our framework – which is why we do not treat it. On the other hand, Wire is more complicated to analyse. Although the core protocol relies on an independent implementation of Signal, its use of cookies and access tokens for authentication and synchronization complicates matters, particularly with respect to powerful adversaries such as insiders. Moreover, the ability to have multiple synchronized devices raises the questions of modelling individual-device compromise and device revocation, for which we would need an extended framework, akin to what is needed to capture MLS security (see below).

A limitation of our approach is that we only consider two-party protocols: as such, even if our *taxonomy* of adversaries is easily extendable to multi-party schemes, such as ART and MLS [5, 14], our *metric* is not. A particular difficulty with extending our framework to multiple parties is the dynamic addition and removal of participants. In two-party schemes, we have two types of evolution, which correspond –roughly– to one, or the other participant's messages. In that case, our metric quantifies the response to the question: if Alice is compromised, after how many of her, and Bob's messages will the channel security heal? However, when we have a dynamically-adaptable set of parties, we would need to account – not only how many turn-switches there are between Alice and non-Alice participants, but also over added and removed users. It is not immediately apparent how best to achieve this, which is why we leave the extension of our metric to multi-party protocols as future work.

## 2 Our PCS metric for SCEKE protocols

Our framework applies to a generalization of two-party secure-channel establishment, which features key-evolution. We call such protocols Secure-Channel Establishment schemes with Key-Evolution (SCEKE), and emphasize that, while post-compromise security (PCS) is particularly relevant to long-lived secure channels, it can also be an attractive property for short-lived channels whose keys evolve from (or depend on) each other.

### 2.1 Definition of SCEKE Protocols

SCEKE schemes allow two parties, Alice and Bob, to initially establish a secure channel (by agreeing on some initial key-material), and then preserve the security of that channel over a long period of time by sequential evolutions of the key material, meant to ensure two properties:

**PFS:** If a user or an instance is fully corrupted at a given moment, all keys established prior to that corruption remain secure;

**PCS:** Even if a user or instance is fully corrupted at some moment, thus breaking channel security, that security will return after a given, finite interval.

Our metric measures the interval required for security to return, with respect to several classes of adversaries (Sec. 2.3). We begin by formalizing the syntax for SCEKE protocols.

**Participants.** We consider schemes featuring participants of two types: parties $P$ making up a set $\mathcal{P}$, and a super-user $\hat{S}$ playing a special part in the protocol (like the registration server used in Signal or the key-derivation center present in identity-based infrastructures). Channel-establishment takes place between two parties (rather than a party and $\hat{S}$), which compute keys and have them evolve.

A SCEKE scheme is initiated by means of a setup algorithm, run by one or more parties, which yield a number of private and public parameters: the super-user $\hat{S}$ is associated with the tuple $(\hat{S}.\mathsf{sk}, \hat{S}.\mathsf{pk})$, while each party $P$ retains identity-bound credentials $(\mathsf{ik}_P, \mathsf{ipk}_P)$. Each of these keys could be a concatenation of credentials, or – if absent – could be void (denoted by $\perp$).

Setup precedes user registration, during which parties $P$ register with the super-user, and $\hat{S}$ builds a database with entries indexed by *unique* party identifiers $P$. The contents associated to each party ID are protocol-specific and is used during secure-channel establishment; exploiting registration leads to a particularly strong type of attack, performed by an *insider* adversary.

**Sessions, instances, and stages.** After registration, parties can run protocol *sessions* with each other. Following [6, 12] formalizations, a protocol *session* occurs between two party-*instances*. The $i$-th instance of $P$ is denoted $\pi_P^i$.

Each instance must run three types of steps during a protocol session: an initial, one-time initialisation step; a recurring message-sending step, occurring every time the instance *sends* an (encrypted) message to its partner; and a recurring receiving step, corresponding to an instance *receiving* (and retrieving) a message from its partner. The sending and receiving steps depend on a crucial notion, that of *stage*.

In SCEKE protocols, messages are encrypted and authenticated before sending, using one or more *message keys*. A stage corresponds to a (protocol-specific) number of messages

[2]See https://signal.org/blog/advanced-ratcheting/.

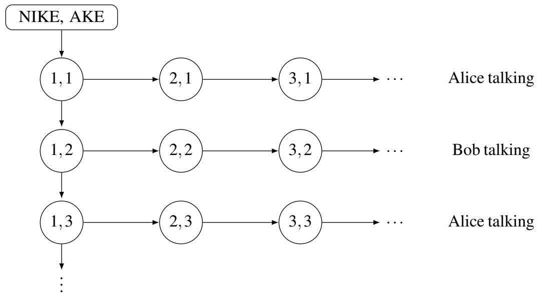


Figure 1: Stage evolution in asynchronous messaging. The protocol initiator begins at stage $(1,1)$ and will continue to send messages along the first horizontal chain. Bob's first reply comes at stage $(1,2)$.

associated with the same message key. When that key evolves, we have moved on to the next stage. The first stage is indexed $(1,1)$ and corresponds to message key $\mathsf{mk}^{1,1}$; more generically, stages are indexed $(x,y)$ with $x,y \geq 1$, with keys evolving through either *horizontal* or *vertical* evolution, as follows.

**Horizontal evolution:** Stage $(x,y)$ turns into $(x+1,y)$. This evolution provides weaker security.

**Vertical evolution:** Stage $(x,y)$ turns into $(1,y+1)$ (we "reset" the x value). This evolution provides stronger security.

As we capture generic key-evolution, our definition of stages is intentionally vague. We do, however, require that honest parties always evolve "forwards", that is, always increasing either the x or the y value. Evolution is depicted in Fig. 1.

**Formalization.** More formally, instance $\pi_P^i$ of parties $P \neq \hat{S}$ keeps track of the following attributes:

$\mathsf{pid}$: partner identifier for the session, denoted $\pi_P^i.\mathsf{pid}$.

$\mathsf{sid}$: session identifier $\pi_P^i.\mathsf{sid}$: an evolving set of instance-specific values.

$\mathsf{stages}$: a list of tuples $(s,v)$, of stages $s=(x,y)$, with values $v \in \{0,1\}$ indicating whether a message was received ($v=1$) or not ($v=0$). By abuse of notation we write $s \in \pi_P^i$ if, and only if, $(s,v) \in \pi_P^i.\mathsf{stages}$.

$Tr$: transcript $\pi_P^i.T$, indexed by stage $s$ describing all data sent or received for this stage. We denote $\pi_P^i.T[s]$.

$\mathsf{rec}$: a list of subsets $\pi_P^i.\mathsf{rec}$, indexed by stage $s$ and indicating messages and metadata received, in order. A special symbol $\perp$ is used for sending stages.

$\mathsf{var}$: a set $\pi_P^i.\mathsf{var}$ of ephemeral values used to compute stage keys, indexed by stage. If a value is used for more than one stage, it will appear under every single stage that it is required for.

**Definition 1 (SCEKE Protocol)** *A Secure-Channel Establishment protocol with Key-Evolution (SCEKE) is a tuple of five algorithms and two interactive protocols:* $\mathtt{SCEKE} = (\mathtt{aSetup}, \mathtt{aKeyGen}, \Pi_{\mathtt{UReg}}, \Pi_{\mathtt{Start}}, \mathtt{aSend}, \mathtt{aReceive}, \mathtt{aRGen})$*:*

$\underline{\mathtt{aSetup}(1^\lambda) \rightarrow (\hat{S}.\mathsf{sk}, \hat{S}.\mathsf{pk}, \mathsf{pparam})}$ *: outputs the public/private long-term keys of super-user $\hat{S}$ and the public system parameters* $\mathsf{pparam}$ *implicitly taken in input by all other algorithms.*

$\underline{\mathtt{aKeyGen}(1^\lambda) \rightarrow (\mathsf{ik}, \mathsf{ipk})}$ *: run by a party $P$ to output public/private long-term credentials* $(\mathsf{ik}, \mathsf{ipk})$*, used at registration (and perhaps further). Either key could be set to a special symbol $\perp$.*

$\underline{\Pi_{\mathtt{UReg}}(P, \hat{S}) \rightarrow (\{\mathsf{sk}, \mathsf{pk}\}, b)}$ *: an interactive protocol run by party $P$ and super- user $\hat{S}$. The latter outputs a bit $b$ (set to* $1$ *for a successful registration), while the former outputs public/private credentials* $(\mathsf{sk}, \mathsf{pk})$ *for $\hat{S}$. The super-user keeps track of a registration database.*

$\underline{\Pi_{\mathtt{Start}}(P, \mathsf{role}, \mathsf{pid}, \hat{S}) \rightarrow (\pi_P^i, b)}$ *: run interactively between $P$ and super-user $\hat{S}$, so as to create an instance of $P$ meant to be talking to an instance of* $\mathsf{pid}$*, such that $P$ has either the role of initiator or responder. If successful, $\hat{S}$ outputs $b$, while $P$ outputs a handle $\pi_P^i$ on its $i$-th instance. Some initial key material might be computed during this phase (like a master secret).*

$\underline{\mathtt{aSend}(\pi_P^i, s, M, AD) \rightarrow (\pi_P^i, C, AD^*) \cup \perp}$ *: on input a state instance $\pi_P^i$, a stage $s$, a message $M$, and associated data $AD$, the algorithm outputs an updated state of the instance $\pi_P^i$, a ciphertext $C$, associated data $AD^*$ or a symbol $\perp$.*

$\underline{\mathtt{aReceive}(\pi_P^i, s, C, AD^*) \rightarrow (\pi_P^i, M, AD) \cup \perp}$ *: on input an instance $\pi_P^i$, a stage $s$, a ciphertext $C$, and associated data $AD^*$, it outputs an updated state of the same instance $\pi_P^i$, a message $M$ and some (possibly transformed) associated data $AD$, or symbol $\perp$.*

$\underline{\mathtt{aRGen}(1^\lambda) \rightarrow (\mathsf{rchk}, \mathsf{Rchpk})}$ *: outputs a public/private keypair used to refresh keys. We call these* ratchet keys*, though they are more generic than the original asymmetric-messaging concept. Either key could be a special symbol $\perp$.*

Protocol correctness relies on matching conversation (intuitively, the partnering of instances running a session).

**Definition 2 (Matching conversation)** *Let* $\mathtt{SCEKE}$ *be a SCEKE protocol, and A, B two parties with instances $\pi_A^i$ and $\pi_B^j$ respectively. $\pi_A^i$ and $\pi_B^j$ have* matching conversation *if and only if* $\pi_A^i.\mathsf{sid} = \pi_B^j.\mathsf{sid}$ *and* $\pi_A^i.\mathsf{pid} = B$ *and* $\pi_B^j.\mathsf{pid} = A$*.*

**A note on out-of-order messages.** A property not considered in our analysis is *out-of-order* (OOO) messaging (or message-loss resilience [4]). Both Signal and SAID provide this feature by design, allowing intermediate messages that are lost to still be recovered in spite of the key having evolved beyond that point. This implies storing several message or chain keys in memory (until they can be used). There are two consequences to our including OOO messaging. First, this implies that even when having matching conversation, Alice and Bob might have non-identical session identifiers (one might have "holes" in the messages that are received). Second, message keys are now computed and stored beyond a single stage. In our formalization, this changes the type of security that is achieved with respect to the values we present in Sec. 3. However, note that capturing message-loss resilience is conceptually compatible with our model.

**Correctness.** If $\pi_A^i$ and $\pi_B^j$ have matching conversation, then a SCEKE protocol `SCEKE` is *correct* if both conditions hold:

- for each stage $s = (x,y)$, both instances have identical $\mathsf{mk}^{x,y}$, and
- $A$ uses `aSend` to output $(\pi_P^i, C, AD^*)$ from $M$ and $B$ inputs $C$ for `aReceive` then $\pi_A^i$ and $\pi_B^j$ are still matching.

## 2.2 Adversarial Model

Our adversary is a Probabilistic Poly-Time adversary $\mathcal{A}$, which manipulates honest parties by using *oracles*. Depending on $\mathcal{A}$'s strength (see the taxonomy in the next section) the attackers may query all, or just some of the oracles presented below.

For reasons that will become apparent when we present our taxonomy, we divide the private keys that parties use during SCEKE sessions into three categories:

**Cross-session Keys:** keys that (intentionally) repeat in at least two sessions[3]. More formally, a key $k$ is cross-session if there exist distinct instances $\pi_P^i$, $\pi_P^j$ of registered party $P$, and (potentially) distinct stages $s \in \pi_P^i$ and $s' \in \pi_P^j$ such that $k \in \pi_P^i\mathsf{var}[s]$ and $k \in \pi_P^j\mathsf{var}[s']$. By definition, the identity and registration keys $\mathsf{ik}_P$ and $\mathsf{sk}$ of $P$ are cross-session. We denote the set of cross-session keys of party $P$ as $P.X\mathsf{sid}$.

**Cross-stage Keys:** keys that (intentionally) repeat in at least two stages of the same session, but not across sessions. More formally, there exists an instance $\pi_P^i$ and distinct stages $s \in \pi_P^i$ and $s' \in \pi_P^i$, such that $k \in \pi_P^i\mathsf{var}[s]$ and $k \in \pi_P^i\mathsf{var}[s']$, but $k \notin P.X\mathsf{sid}$. We denote the set of cross-stage keys belonging to instance $\pi_P^i$ as $\pi_P^i.X\mathsf{stage}$.

**Stage-specific Keys:** keys occurring in only one stage of one protocol instance $\pi_P^i$, *i.e.*, $k \in \pi_P^i.\mathsf{var}[s]$ for some stage $s$, but $k \notin (P.X\mathsf{sid} \bigcup \pi_P^i.X\mathsf{stage})$. We denote by $\pi_P^i.\mathtt{1}\mathsf{stage}$ the set of all stage-specific keys of instance $\pi_P^i$.

**Oracles.** The adversary can register malicious users; compromise users to reveal, respectively, cross-session, cross-stage, and stage-specific private values; and manipulate communication by instantiating new sessions and sending/receiving messages. The adversary's goal will be distinguished from random a target message-key that is freshly and honestly generated. Thus, each instance needs to also store the attribute $\pi_P^i.\mathsf{b}[s]$: a challenge bit randomly chosen for each instance for stage $s$. If $b = 1$, the output is the real message key, else the output is a random key.

We describe the PCS-game in Sec. 2.4, while a taxonomy of adversarial *types* are introduced in Sec. 2.3. In the game, the adversary may query (a subset of) the following oracles:

[3] We thus formally exclude collisions in randomness

$\underline{\mathsf{oUReg}(P)}$: runs `aKeyGen` on party $P$ *i.e.*, $\mathcal{A}$ can register malicious $P$ to an honest $\hat{S}$.

$\underline{\mathsf{oStart}(P, \mathsf{role}, \mathsf{pid}, hon)}$: runs $\Pi_{\mathtt{Start}}$ to create a new instance of an existing honest party with the role role and intended partner pid. The added value $hon$ is a bit, which, if set to 1, runs the protocol with the challenger posing as $\hat{S}$, whereas if $hon = 0$, the protocol is run with the adversary posing as $\hat{S}$.

$\underline{\mathsf{oTest}_b(\pi_P^i, s)}$: for honest parties, valid instances, valid stages, and a computed message-key at stage $s$, returns that key (if $\pi_P^i.\mathsf{b}[s] = 1$) or a random key of the same length ($\pi_P^i.\mathsf{b}[s] = 0$). This oracle can only be queried once.

$\underline{\mathsf{oSend}(\pi_P^i, s, AD)}$: two modes for this oracle: honest or maliciously-controlled. For $AD = \perp$ (other values are valid), $\pi_P^i$ generates new key pair using `aRGen` for stage $s$ then it runs `aSend`, and outputs the additional data. Otherwise, the oracle simulates the sending algorithm with adversarially-chosen $AD$.

$\underline{\mathsf{oReceive}(\pi_P^i, s, AD)}$: oracle also in two modes. In honest mode, $AD$ is valid since output by oSend at stage $s$ by $\pi_P^i$'s partner. For the adversarial mode, $AD$ is always considered correct (allowing communication hijacking for instance).

$\underline{\mathsf{oReveal.XSid}(P)}$: corrupts $P$, giving $\mathcal{A}$ access to $P.X\mathsf{sid}$.

$\underline{\mathsf{oReveal.XStage}(\pi_P^i, s)}$: for stage $s$, it leaks the set of keys $\pi_P^i.X\mathsf{stage} \bigcap \pi_P^i\mathsf{var}[s]$ of cross-stage values.

$\underline{\mathsf{oReveal.1Stage}(\pi_P^i, s)}$: for stage $s$, it leaks the set $\pi_P^i.\mathtt{1}\mathsf{stage} \bigcap \pi_P^i\mathsf{var}[s]$ of stage-specific values.

Like in [7], $\mathcal{A}$ does not have access to the real ciphertext, which is a trivial distinguisher.

## 2.3 A taxonomy of adversaries

We classify adversaries in terms of: reach; power; and access, as discussed below. Although the security games and winning conditions are mostly equivalent, different adversaries will learn a different subset of values upon compromising a party, and will be allowed different sequences of oracle queries.

**Reach.** Our model features three types of corruption oracles: oReveal.XSid, oReveal.XStage, and oReveal.1Stage, revealing, respectively, the party's cross-session (long-term) keys, cross-stage keys, and stage-specific keys. Of these, the latter are assumed to be the least protected because as they are the least impactful during key-evolution. We distinguish the following adversarial classes:

- <u>Local adversaries:</u> are only allowed access to the oReveal.1Stage oracle;
- <u>Medium adversaries:</u> may query both oReveal.1Stage and oReveal.XStage, but not oReveal.XSid;
- <u>Global adversaries:</u> may query all three oracles.

**Power.** We distinguish between attackers which extract information from honest participants via their reveal oracles, and stronger adversaries, which extract data and then use it to hijack honest sessions, or for other (evil) purposes. This reasoning leads to a classification between:

- Active adversaries: The attacker may use the malicious modes of the oSend and oReceive oracles on the target instance $\pi_P^i$, or on the instance it has matching conversation with. We define below one potential strategy of such attackers, namely session hijacking, but active adversaries are not restricted to only it. In short, (successfully) hijacking a session enables the adversary to insert its own key material and increase the interval required before healing (or make the channel unable to heal at all);
- Passive adversaries: These attackers may not use the malicious modes of the sending and receiving oracles on the target instance, nor its partner.

We define the *hijacking* of a session run between $\pi_A^i$ and its partner $\pi_B^j$ at some stage $s_h = (x_h, y_h)$ (for which we assume w.l.o.g. that $A$ is the sender) the event that the following conditions hold simultaneously:

1. $\mathcal{A}$ has queried $\mathsf{oReceive}(\pi_B^j, s_h, AD_h)$;
2. $AD_h$ were never output by an $\mathsf{oSend}(\pi_A^i, s_h, \cdot)$ query;
3. there exists a value $v \in AD_h$, but such that $v \notin \pi_A^i.\mathsf{var}[s_h] \cup \pi_B^j.\mathsf{var}[s_h]$.

We call stage $s_h$ *successfully hijacked* if in addition the oReceive query in 1. yielded an output different from $\perp$.

**Access.** The last criterion in our taxonomy is access. Typically, channel-security is defined with respect to a Person-in-the-Middle attacker. However, some such protocols also feature a centralized entity with more extensive access and thus greater potential to wreak havoc – in our framework, the super-user $\hat{S}$. We divide attackers into two categories:

- Insider adversaries: they *are* in fact the super-user. Throughout the game, they receive from the challenger all the private keys and database information amassed by $\hat{S}$.
- Outsider adversaries: these attackers do not receive any $\hat{S}$ data. Since additionally $\mathcal{A}$ has no oracle-access to corrupting $\hat{S}$, the latter will remain honest.

**Adversarial types.** We consider adversaries whose types are a composition of three characteristics, in the order (power, reach, access). The weakest adversary is a passive local outsider. The strongest is an active global insider. All other characteristics being equal, active attackers are stronger than passive ones; also, global attackers are stronger than medium ones, which are in turn stronger than local ones; finally, insiders are stronger than an outsiders.

Nevertheless, intermediate adversaries with more than two varying characteristics are not as easy to compare. This is particularly the case for insider attacks, for which the information obtained by the insider is highly protocol-specific. The same holds for active local adversaries versus passive global ones. In our case, moreover, comparing such adversaries asymptotically is not as interesting as quantifying, for each adversary, the exact healing speed of the scheme. In Fig. 2, we recap the adversary's access to oracle depending on its type.

| YYY | LPO | MPO | GPO | LPI | MPI | GPI | LAO | MAO | GAO | LAI | MAI | GAI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1Stage | ✓ | | | ✓ | | | ✓ | | | ✓ | | |
| XStage | | ✓ | | | ✓ | | | ✓ | | | ✓ | |
| XSession | | | ✓ | | | ✓ | | | ✓ | | | ✓ |
| Access $\hat{S}$.sk | | | | ✓ | ✓ | ✓ | | | | ✓ | ✓ | ✓ |
| oReceive | H | H | H | H | H | H | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| oSend | H | H | H | H | H | H | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| oUReg | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| oStart | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Figure 2: Available oracles depending on type, labelled reach‖power‖access. For instance, LAO denotes Local Active Outsider adversary. We omit oTest oracle since all adversaries may query it. H denotes an honest call to the oracle.

## 2.4 A metric for PCS

The adversary $\mathcal{A}$ plays against a challenger $\mathcal{C}$ in the following security game $\mathsf{Exp}^{PCS}_{\mathsf{SCEKE}}(\lambda, \mathcal{A})$, which is also depicted in Fig. 3:

- $\mathcal{C}$ runs `aSetup` and forwards all the public values to $\mathcal{A}$. $\mathcal{C}$ also simulates the registration of all the honest parties.
- $\mathcal{A}$ has access to algorithms `aKeyGen` and `aRGen` and, depending on its type, may adaptively query a subset of these oracles (see also Fig. 2):
  - $\mathsf{oUReg}(P)$ (all attackers);
  - $\mathsf{oStart}(P, \mathsf{role}, \mathsf{pid}, 1)$ (outsider $\mathcal{A}$) and $\mathsf{oStart}(P, \mathsf{role}, \mathsf{pid}, 0)$ for (insiders);
  - $\mathsf{oSend}(\pi_P^i, s, \perp)$ (passive $\mathcal{A}$) and $\mathsf{oSend}(\pi_P^i, s, AD)$ (active $\mathcal{A}$);
  - $\mathsf{oReceive}(\pi_P^i, s, \perp)$ (passive $\mathcal{A}$) and $\mathsf{oReceive}(\pi_P^i, s, AD)$ (active $\mathcal{A}$);
  - $\mathsf{oReveal.XSid}(P)$ (global $\mathcal{A}$);
  - $\mathsf{oReveal.XStage}(\pi_P^i, s)$ (medium $\mathcal{A}$);
  - $\mathsf{oReveal.1Stage}(\pi_P^i, s)$ (local $\mathcal{A}$).
- At some point, $\mathcal{A}$ outputs a party instance $\pi_P^\star$ and a stage $s^\star = (x^\star, y^\star)$. The challenger $\mathcal{C}$ runs $\mathsf{oTest}_b(\pi_P^\star, s^\star)$ and outputs the true $\pi_P^\star.\mathsf{mk}^s$ or a random key.
- The attacker may continue to use its oracles/algorithms, until it outputs a final bit $d$.

We say that $\mathcal{A}$ wins $\mathsf{Exp}^{PCS}_{\mathsf{SCEKE}}(\lambda, \mathcal{A})$ if and only if $d = \pi_P^\star.\mathsf{b}[s^\star]$, and if the winning conditions below hold. The *advantage* of the adversary is computed as: $|\Pr[\mathcal{A} \text{ wins } \mathsf{Exp}^{PCS}_{\mathsf{SCEKE}}(\lambda, \mathcal{A})] - \frac{1}{2}|$.

**Further winning conditions.** In order to win the $\mathsf{Exp}^{PCS}_{\mathsf{SCEKE}}(\lambda, \mathcal{A})$ game, $\mathcal{A}$ must guess the real-or-random bit $b$ for the target message key, and must do so by a non-trivial attack (for instance, it would be trivial to win by revealing the target message key, and then attempting to distinguish it). Attacks are classified as trivial or non-trivial depending on adversary type. We express them as a conjunction of predicates parametrized by $\mathcal{A}$'s type and resulting PCS security.

```
$\underline{\mathsf{Exp}^{PCS}_{\mathsf{SCEKE}}(\lambda, \mathcal{A})}$
$(\hat{S}.\mathsf{sk}, \hat{S}.\mathsf{pk}, \mathsf{pparam}) \leftarrow \mathcal{C}^{\mathtt{aSetup}(1^\lambda)}$
$(\mathcal{P} = \{P_1, \cdots P_{\mathsf{n}_\mathcal{P}}\}) \leftarrow \mathcal{C}(\lambda, \mathsf{n}_\mathcal{P})$
$(\mathsf{ik}_i, \mathsf{ipk}_i) \leftarrow \mathcal{C}^{\mathtt{aKeyGen}(1^\lambda)} \quad \forall i \in \{1, \cdots, \mathsf{n}_\mathcal{P}\}$
$\mathcal{O}_{\mathsf{type}} \leftarrow \left\{ \begin{array}{l} \mathsf{oUReg}(\cdot), \mathsf{oStart}(\cdot,\cdot,\cdot,\cdot), \mathsf{oReveal}[\mathcal{A}.\mathsf{reach}](\cdot,\cdot), \mathsf{oSend}(\cdot,\cdot,\cdot,\cdot), \\ \mathsf{oReceive}(\cdot,\cdot,\cdot,\cdot),, \mathcal{RO}_1(\cdot), \mathcal{RO}_2(\cdot) \end{array} \right\};$
$(\pi^\star_P, s^\star) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{type}}}(1^\lambda)$
$K \leftarrow \mathsf{oTest}_{b^\star}(\pi^\star_P, s^\star)$
$d \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{type}}}(\lambda, \mathsf{n}_\mathcal{P}, K)$
$\boxed{\mathcal{A} \text{ wins iff. } d = b^\star \text{ and } (\neg\mathsf{oUReg}(P) \vee \neg\mathsf{oUReg}(\pi^i_P.\mathsf{pid})) = \top}$
```

Figure 3: The PCS game $\mathsf{Exp}^{PCS}_{\mathsf{SCEKE}}(\lambda, \mathcal{A})$ between adversary $\mathcal{A}$ and challenger $\mathcal{C}$, parametrized by the security parameter $\lambda$ and number of *honest* parties $\mathsf{n}_\mathcal{P}$. $\mathcal{A}$ can query a set of oracles $\mathcal{O}_{\mathsf{type}}$, subject to type. We denote by $\mathsf{oReveal}[\mathcal{A}.\mathsf{reach}]$ the precise reveal oracle allowed to $\mathcal{A}$, subject to its reach (local, medium, or global).

**Definition 3 (($\chi, \Upsilon$)-PCS security)** *A* SCEKE *protocol is* $(\chi, \Upsilon)$*-PCS-secure against an adversary* $\mathcal{A}$*, for* $\chi, \Upsilon \in \mathbb{N}$ *and* $\mathcal{A}$ *of one of the* 12 *types above if, and only if, assuming* oTest *will be queried for instance* $\pi^i_P$*, the last stage for which* $\mathcal{A}$ *queried* oReveal.XStage *or* oReveal.1Stage *for either* $\pi^i_P$ *or its matching instance is* $s^* = (x^*, y^*)$ *and the following conditions hold:*

- *The adversary has a non-negligible advantage to win the game* $\mathsf{Exp}^{PCS}_{SCEKE}(\lambda, \mathcal{A})$ *when querying* oTest *for* $s_{\mathsf{Test}} = (x_{\mathsf{Test}}, y_{\mathsf{Test}})$ *such that:*
  - *If* $\Upsilon = 0$*,* $x_{\mathsf{Test}} < x^* + \chi$ *and* $y_{\mathsf{Test}} = y^*$*;*
  - *If* $\Upsilon > 0$*,* $x_{\mathsf{Test}}$ *is arbitrary and* $y_{\mathsf{Test}} < y^* + \Upsilon$*.*

  *If, moreover, the adversary is allowed to query* oReveal.XSid*, then* $\mathcal{A}$ *has a non-negligible chance to win for all instances of party* $P$ *which are not yet instantiated, or have not yet reached stage* $s = (x, y)$ *such that:*
  - *If* $\Upsilon = 0$*, then* $x \geq \chi$ *and* $y \geq 1$*;*
  - *If* $\Upsilon > 0$*, then* $x > 1$ *and* $y \geq \Upsilon$*.*
- *The adversary has a neglibile advantage to win if* oTest *is queried for* $s_{\mathsf{Test}}$ *other than those specified in the first bullet point.*

*We allow both* $\chi$ *and* $\Upsilon$ *to take a special value* $\infty$*, which corresponds to "an arbitrary number of stages" obtained through horizontal and respectively through vertical evolution.*

## 3 Case Studies

We apply our metric to 3 use cases: the PKI-based messaging protocol Signal, the Identity-Based messaging protocol SAID, and the suite of mobile 5G Handover protocols. Although seemingly very different, they all can be modelled as SCEKE schemes, which shows the generality of our framework.

### 3.1 Signal as a SCEKE protocol

Signal is a natural instantiation of SCEKE protocols. Like most asynchronous-messaging schemes, Signal conversations are turn-based, between speakers who need not be online simultaneously. Each message corresponds to one stage, *i.e.*, each message-key is used only once. New keys are generated in two different ways: when the person who was speaking sends a new message, we have a horizontal evolution; when the speaker changes, we have a vertical evolution.

Signal also features a natural super-user: a centralized credential server storing user public keys. Our SCEKE framework includes security with respect to such powerful insiders, an aspect often overlooked by prior work [13].

We first compare our model with the one by Cohn-Gordon *et al.* [13]; then we model Signal as a SCEKE protocol. The detailed description of the protocol is provided in Appendix A. Then we quantify PCS-security with respect to all the adversaries in Sec. 2.3. The security proofs are given in Appendix C.

| | Outsider | Reach | Signal | SAID | *5G-SCEKE* | *5G-SCEKE*$^+$ |
|---|---|---|---|---|---|---|
| Access | Passive | Global | $(\infty, 2)$ | $(\infty, 2)$ | $(\infty, \infty)$ | $(1, 0)$ |
| | | Medium | $(\infty, 2)$ | $(\infty, 2)$ | $(\infty, \infty)$ | $(1, 0)$ |
| | | Local | $(\infty, 1)$ | $(1, 0)$ | $(\infty, 1)$ | $(1, 0)$ |
| | Active | Global | $(\infty, \infty)$ | $(\infty, \infty)$ | $(\infty, \infty)$ | $(\infty, \infty)$ |
| | | Medium | $(\infty, \infty)$ | $(\infty, \infty)$ | $(\infty, \infty)$ | $(\infty, \infty)$ |
| | | Local | $(\infty, 1)$ | $(1, 0)$ | $(\infty, 1)$ | $(1, 0)$ |
| | **Insider** | **Reach** | **Signal** | **SAID** | ***5G-SCEKE*** | ***5G-SCEKE*$^+$** |
| Access | Passive | Global | $(\infty, 2)$ | $(\infty, 2)$ | $(\infty, \infty)$ | $(\infty, \infty)$ |
| | | Medium | $(\infty, 2)$ | $(\infty, 2)$ | $(\infty, \infty)$ | $(\infty, \infty)$ |
| | | Local | $(\infty, 1)$ | $(\infty, 1)$ | $(\infty, \infty)$ | $(\infty, \infty)$ |
| | Active | Global | $(\infty, \infty)$ | $(\infty, \infty)$ | $(\infty, \infty)$ | $(\infty, \infty)$ |
| | | Medium | $(\infty, \infty)$ | $(\infty, \infty)$ | $(\infty, \infty)$ | $(\infty, \infty)$ |
| | | Local | $(\infty, \infty)$ | $(\infty, \infty)$ | $(\infty, \infty)$ | $(\infty, \infty)$ |

Figure 4: Results for our metric on PCS-security for Signal, SAID, 5G handover and its variant denoted Ext-*5G-SCEKE*.

**Comparing security models.** Our framework can be seen, in many ways, as a generalization of Cohn-Gordon *et al.*'s Signal-specific security model [13]. They described a real-or-random key-indistinguishability experiment akin to ours, for which the Person-in-the-Middle adversary $\mathcal{A}$ can test stages freely in order to distinguish their message-keys from random. $\mathcal{A}$ wins assuming that it guesses correctly and that a given freshness predicate holds.

We begin by stating that the adversary described by [13] is a passive outsider: they rule out adversarial interventions within the target session, and do not consider security with respect to the super-user. Finally, the oracles they consider are slightly different from ours, as we describe below.

From Fig. 5, we can infer that:

$$\mathsf{oReveal.1Stage} \implies \mathsf{RevSessKey} \wedge \mathsf{RevRand} \wedge \mathsf{RevStateMiddle}$$
$$\mathsf{oReveal.XStage} \implies \mathsf{RevRand} \wedge \mathsf{RevStateInit}$$
$$\mathsf{oReveal.XSid} \implies \mathsf{RevLongTermKey} \wedge \mathsf{RevRand}$$

Thus the adversaries captured in [13] can adopt more fine-grained strategies than ours. For instance, in our model, if the adversary wants a particular cross-stage key, it essentially will receive *all* such keys. As a consequence, we lose the ability to rank, say, cross-stage keys in terms of how dangerous they are to healing. Yet, (instantiations of) the predicates

| | ms | ephk | ck | mk | rk | rchk | ik | prek |
|---|---|---|---|---|---|---|---|---|
| oReveal.1Stage | ✓ | ✓ | ✓ | ✓ | | | | |
| oReveal.XStage | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| oReveal.XSid | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| RevSessKey | | | | ✓ | | | | |
| RevLongTermKey | | | | | | | ✓ | |
| RevMedTermKey | | | | | | | | ✓ |
| RevRand | | ✓ | | | | ✓ | | |
| RevStateInit | | | | | ✓ | | | |
| RevStateMiddle | | | ✓ | | | | | |

Figure 5: Revealed keys per oracle queries: ✓s indicate revealed keys. The 3 upper rows list oracles in our model, while the bottom ones are oracles from [13]. Notice that for Signal, we split oracle RevState into RevStateInit (which can be used only at the beginning of a stage-chain) and RevStateMiddle (for queries inside a chain *i.e.*, $x > 1$).

described above are in fact also found amongst the winning conditions of [13], signifying that the same

Yet, in reality (as described in the proofs), the winning predicates of [13] imply that the adversary does not essentially benefit from the additional freedom given by those fine-grained queries. Thus, while our two frameworks are syntactically incomparable, they are akin in spirit. In addition, our model allows us to account for additional adversary types, including active adversaries and insiders.

**Protocol description.** Let $\mathcal{P}$ be a set of honest users (with unique identities). Our super-user $\hat{S}$ is a centralized PKI server.

SETUP. During the global setup of the protocol, the super-user $\hat{S}$ chooses a DH-based signature algorithm, hash functions and KDFs, and a secure-channel establishment protocol required at registration (such as TLS)[4]. Then, $\hat{S}$ generates keys required for its authentication in the secure-channel establishment, notably $(\hat{S}.\mathsf{sk}, \hat{S}.\mathsf{pk})$. We also assume that each user in $\mathcal{P}$ has keys allowing it to register over a secure channel to $\hat{S}$, but make no assumptions as to their structure; we merely require that they provide secure authentication. All the algorithms and $\hat{S}.\mathsf{pk}$ are part of the public system parameters.

KEY GENERATION. During key-generation, each party generates signature identity keys $(\mathsf{ik}_P, \mathsf{ipk}_P)$ for the signature scheme chosen at setup.

USER REGISTRATION. Over an AKE-secure channel between $\hat{S}$ and each user $P$, the latter registers a key-bundle consisting of: a long-term identity key $\mathsf{ipk}_P$, a medium-term key $\mathsf{prepk}_P$ signed with $\mathsf{ik}_P$, and optional ephemeral *public* keys $\mathsf{ephpk}_P$. Both $\mathsf{ipk}_P$ and $\mathsf{prepk}_P$ are used across multiple sessions, whereas each ephemeral public key $\mathsf{ephpk}_P$ is only used in one session and then removed from the server. Subsequent calls to this algorithm allow users to update key material they have previously submitted. We stress that the server is never

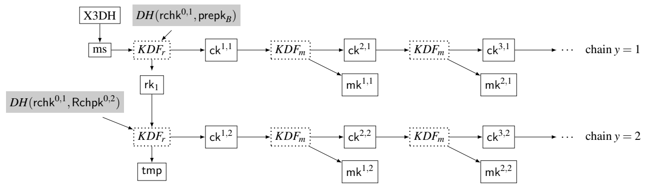


Figure 6: Signal's key schedule, in which vertical evolutions are boxed in grey. Horizontal evolutions are along chains represented horizontally.

given the user's private keys.

INSTANCE INITIALISATION. Alice (the initiator) begins a session with Bob by querying the semi-trusted server, over an authenticated channel, for Bob's credentials, which allows Alice to establish the master secret ms.

The master secret will yield an intermediate *root key* $\mathsf{rk}_1$ and the first *chain key* $\mathsf{ck}^{1,1}$; the latter will be input to a key-derivation function (KDF) in order to output a new key $\mathsf{ck}^{2,1}$ and the first *message key* $\mathsf{mk}^{1,1}$, which will be used to authenticate and encrypt Alice's first message to Bob, corresponding to stage $(1,1)$ of the session.

SENDING AND RECEIVING. For the remainder of the session, Alice and Bob exchange encrypted messages. On stages with odd $y$, Alice is the sender and Bob is the receiver, while on stages with even $y$, it is the other way around. In each stage, corresponding to a single encrypted message, the included metadata allows that message's receiver to make his keys evolve, either horizontally (it receives and decrypts a new message) or vertically (the receiver decides to start talking).

Fig. 12 gives this key-derivation process.

The associated metadata at stage $(x,y)$ consists of the identities of the two speakers, a ratchet public key $\mathsf{Rchpk}^y$ (usable for vertical evolution), and the index $x$ of the message. Exceptionally, for messages sent at stages $(\cdot,1)$, the metadata must also include the public key $\mathsf{Epk}_A$ corresponding to Alice's private key $\mathsf{ek}_A$ used during session initialisation. This metadata is sent as Associated Authenticated Data (AAD) within each AEAD-encrypted ciphertext[5]. Thus, this data passes in clear, but is authenticated as part of the ciphertext.

**The PCS-security of Signal.** We begin the analysis by splitting the key material used in Signal into stage-specific, cross-stage, and cross-session keys.

The keys used in Signal for a single stage only are: the message[6] keys $\mathsf{mk}^{x,y}$, the chain keys $\mathsf{ck}^{x,y}$, and also a particular key used only at stage $(1,1)$, namely ephk.

On the other hand, private ratchet keys $\mathsf{rchk}^{x,y}$ and the keys used at the roots of each chain(denoted $\mathsf{rk}_y$ for odd $y$ and $tmp^y$ for even $y$) are stored throughout the existence of the chain, until the next vertical evolution. In other words, they are cross-

[4]In other words, we assume that whenever they upload keys to the server, parties do so over a mutually-authenticated secure channel (with standard AKE security).

[5]AEAD stands for Authenticated Encryption with Associated Data.

[6]We explicitly do not consider the fact that in Signal keys can actually be precomputed in the case of out-of-order arrivals, since this is not the most frequent way in which the protocol is used.

| Keys | Signal | SAID | *5G-SCEKE* or *5G-SCEKE$^+$* |
|---|---|---|---|
| Cross-Session | ik,prek | ik, ID.sk,IBS.sk | K |
| Cross-Stage | rk,rchk,*tmp* | ms, rk,rchk | $\mathsf{K}_{AMF} = \mathsf{rk}$ |
| Single-Stage | ephk, ms,mk, ck | mk,ck, *r* | rchk =v , $\mathsf{K}_{AS} = \mathsf{mk}$, $\mathsf{K}_{gNB} = \mathsf{ck}$ |

Table 1: Taxonomy on keys used in Signal, SAID and our 5G handover procedures model.

stage keys. We give a summary of the key material in Table 1.

**Theorem 1** *Consider the Signal protocol modelled as a SCEKE scheme, as presented above. The following results hold in the random oracle model (by replacing the KDFs with random oracles), under the Gap Diffie-Hellman assumption, and assuming the AKE security of the channels established between honest users and an honest $\hat{S}$:*

- *Signal is $(\infty, 1)$-PCS secure against: local outsiders (passive and active), local passive insiders;*
- *Signal is $(\infty, 2)$-PCS secure against: medium passive adversaries (outsiders and insiders), and global passive attackers (outsiders and insiders);*
- *For all other adversaries, Signal is $(\infty, \infty)$-PCS secure.*

Note that the results are also systematized in Fig. 4. The proofs of this theorem consist of two types of statements: first, we need to show an attack for the stages that are vulnerable to the attacker, then we need to prove that beyond those stages, security holds. The second parts of the proofs can be found in Appendix C, but we briefly indicate the attacks providing the first part of the proofs below.

LOCAL PASSIVE OUTSIDER Here the security loss is a result of the symmetric ratchets: once $\mathsf{ck}^{x,y}$ is compromised, $\mathcal{A}$ learns all the chain and message keys derived symmetrically from it. The ratchet key $\mathsf{rchk}^y$ is not amongst the data revealed through oReveal.1Stage. When it is used in input at stage (1,y+1), $\mathcal{A}$ can no longer compute keys derived from this key.

MEDIUM PASSIVE OUTSIDER As opposed to the previous case, the attacker can now query oReveal.XStage and learn ratchet keys rchk, and root keys. Knowledge of the ratchet key $\mathsf{rchk}^y$ allows $\mathcal{A}$ to compute $\mathsf{DH}^{0,y+1} = DH(\mathsf{Rchpk}^y, \mathsf{Rchpk}^{y+1})$ at the beginning of chain $y+1$ and derive all the keys in chain $y+1$ (hence implying $(\infty, 2)$-PCS security). Fortunately, this stops at stage $(1, y+2)$, since $\mathcal{A}$ cannot use $\mathsf{rchk}^y$ to compute $\mathsf{DH}^{0,y+2}$, thus giving the PCS bound.

GLOBAL PASSIVE OUTSIDER The adversary's access to oReveal.XSid provides user identity keys and pre-keys. However, these values cannot help a passive adversary beyond learning the master secret ms (via oReveal.1Stage). This essentially reduces a global passive outside to a medium passive adversary.

LOCAL ACTIVE OUTSIDERS In this weakest form of active outsider attacks, the attacker can still actively *use* the information captured through corruption, in addition to learning it by compromising either endpoint. Unfortunately, this is not helpful, since in order to go beyond the $(\infty, 1)$ bound provided in the local passive outsider case, $\mathcal{A}$ would require knowing the chain's current root key. Although this is a Denial of Service (DoS) attack, it will not affect PCS security.

OTHER ACTIVE OUTSIDERS The attacker has access to oReveal.XStage, and so to the root key it was missing in the previous cases. As a result, the attacker can use its active capacity to learn the message and chain keys of some stage $(1, y)$ and then *use them* to inject its own ratcheting information, towards the receiver at chain $y$. Then, by using the root key (via oReveal.XStage) it keeps up with all future ratchets. This compromises all the future keys in these sessions, yielding an $(\infty, \infty)$-PCS security.

INSIDER PASSIVE ATTACKS The knowledge of the super-user's private key $\hat{S}$.sk will not help the adversary beyond an outsider adversary's capacity. This is the situation that corresponds to an honest-but-curious server – for which Cohn-Gordon *et al.* considered (and proved) the security we also explained for the outsider case. This explains why we have the same bounds for the insider and outsider passive adversaries.

INSIDER ACTIVE ATTACKS At the opposite end of the scale are active insider attacks, which basically capture a fully malicious centralized server. At user registration, the malicious super-user behaves as normal. However, at session setup, when Alice wants to talk to Bob, $\hat{S}$ forwards a key-bundle of its own making, to which it has the corresponding private keys. The attacker then does the same when Bob asks for Alice's credentials (forwarding keys from the same bundles), thus ensuring that it can run a Person-in-the-Middle attack between the two users. This type of attack requires no reveal queries on any of the user key material – hence, Signal provides $(\infty, \infty)$-PCS security (no healing at all) for all insider active attackers.

**Signal with acknowledgements.** More recent implementations of Signal have slightly evolved from the core protocol we described in this paper, and have added an acknowledgement, which essentially reduces message-chain length to 1. In addition, root and ratchet keys become stage-local keys, thus augmenting security against local adversaries to $(\infty, 2)$.

**Signal with two-factor authentication.** A way to reduce the impact of insider attacks is to have users verify the identity keys of other users prior to instantiating sessions with them – a type of two-factor authentication. However, such verifications are not without dangers, as described in recent literature [16].

## 3.2 SAID as a SCEKE protocol

Introduced in 2019, SAID's main aim is to strengthen authentication in messaging protocols [7]. The protocol is proposed in the identity-based (IB) setting, requires a Key-Derivation Center to replace Signal's credential server, and makes substantial modifications to key-evolution. We briefly review

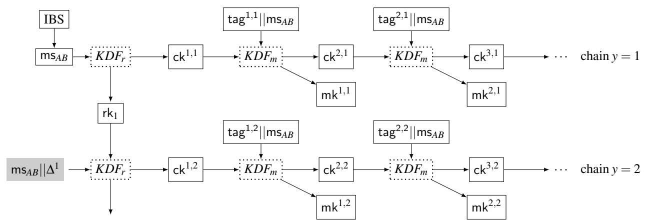


Figure 7: The key schedule of SAID.

SAID here, and include more details in Appendix B.

**Protocol description.** Our description below follows that of [7], but expands upon the user-registration part. We notably consider additional keypairs for the KDC and protocol participants – which will allow them to establish the secure channel they require at registration. Although these keys do not feature in SAID, Blazy *et al.* [7] do suppose that a mutually-secure channel exists during that process.

SETUP. SAID relies on an identity-based signature scheme $\mathsf{IBSig} = (\mathtt{aIBS.Setup}, \mathtt{aIBS.Extract}, \mathtt{aIBS.Sign}, \mathtt{aIBS.Vfy})$ and a type-3 pairing $e$. At system setup the KDC generates global public and private parameters. It must notably generate global setup values $(\mathsf{IBS.msk}, \mathsf{IBS.mpk})$ for the IB signature scheme and parameters $\mathsf{ID.msk} \xleftarrow{\$} \mathbb{Z}_p$ (private) and $\mathsf{ID.mpk} = g_2^{\mathsf{ID.msk}} \in \mathbb{G}_2$ (public) for embedding identities into private identity keys. $\hat{S}$ generates a key-pair that enables authentication in the AKE protocol of its choice (*e.g.*, TLS 1.3), denoting them $(\hat{S}.\mathsf{sk}, \hat{S}.\mathsf{pk})$, then appends $\mathsf{ID.msk}$ and $\mathsf{IBS.msk}$ to $\hat{S}.\mathsf{sk}$.

KEY GENERATION. This step proceeds as in [7], but we additionally have parties register some non-IB keypairs $(\mathsf{ik}, \mathsf{ipk})$, usable during registration.

REGISTRATION. Users $P$ register over a secure channel established with the KDC ($P$ uses its $(\mathsf{ik}, \mathsf{ipk})$ tuple and KDC, $(\hat{S}.\mathsf{sk}, \hat{S}.\mathsf{pk})$). Over this channel, $P$ sends her identity $P$ (*e.g.*, a phone number, email address, etc.), to the KDC. The KDC returns the user's secret signing key $\mathsf{IBS.sk}_P \leftarrow \mathtt{aIBS.Extract}(\mathsf{IBS.ppar}, \mathsf{IBS.msk}, P)$ and secret identification key $\mathsf{ID.sk}_P = H_2(P)^{\mathsf{ID.msk}}$. Thus, the KDC knows all the users' private keys.

INSTANCE INITIALISATION. In SAID, instance-initialisation requires no user-KDC interaction (thus we deem $\hat{S}$'s contribution void). Initiator Alice will choose randomness $r$ and compute $\mathsf{ms}_{AB} = e(H(B), \mathsf{ID.mpk})^r$: in other words, Alice embeds the Bob's identity into the master secret. Alice also generates a random *tag*: $\mathsf{tag}^{1,1}$, and uses it and the master secret to derive the root key $\mathsf{rk}_1$ and the first chain key $\mathsf{ck}^{1,1}$. The use of fresh tags is specific to SAID and ensures that keys are unlikely to repeat. A KDF is used to derive the chain key $\mathsf{ck}^{2,1}$ and first message key $\mathsf{mk}^{1,1}$ from $\mathsf{ms}_{AB}$ and $\mathsf{ck}^{1,1}$.

Unlike Signal, in SAID the master secret $\mathsf{ms}_{AB}$ is used at every stage; thus Bob has to regularly prove knowledge of his private identity-key, and Alice, of the secret $r$ signed with her IB signing key. In [7] all parties ($P$) store values $\mathsf{ik}_P$ and master secrets $\mathsf{ms}_{P*}$ of started sessions, and $\mathsf{ms}_{*P}$ of responded sessions in a *trusted execution environment* – which we abstract.

SENDING AND RECEIVING. Stages and keys evolve in SAID similarly to Signal:

- **Symmetric ratcheting:** To go from stage $(x,y)$ to $(x+1,y)$, the current speaker generates a new tag $\mathsf{tag}^{x+1,y}$ to be input with chain key $\mathsf{ck}^{x+1,y}$, in order to output $\mathsf{ck}^{x+2,y}$ and $\mathsf{mk}^{x+1,y}$. The two precise substeps are detailed in Appendix B.
- **Asymmetric ratcheting:** When speakers change, the key material is freshened up with Diffie-Hellman randomness: on input the master secret, a value $\Delta^{y-1} = DH(\mathsf{Rchpk}^{0,y-1}, \mathsf{Rchpk}^{0,y})$, and the root key $\mathsf{rk}_y$, a KDF outputs $\mathsf{rk}_{y+1}$ and $\mathsf{ck}^{1,y+1}$. The chain key, master secret, and a fresh tag $\mathsf{tag}^{1,y+1}$ are used to obtain the chain's first message key $\mathsf{mk}^{1,y+1}$.

The receiver gets the public key material allowing it to ratchet correctly as authenticated metadata. For the first message chain, Alice sends the following AAD: the public value $h = g_2^r$ corresponding to the secret $r$ that the initiator used to compute the master secret; the stage's horizontal index $x$; a fresh public ratchet key $\mathsf{Rchpk}^1 = g_1^{\mathsf{rchk}^1}$; the tag of the current message; the user identities; and a signature over everything except the tag: $\sigma \leftarrow \mathtt{aIBS.Sign}(\mathsf{IBS.ppar}, \mathsf{IBS.sk}_A, \{\mathsf{meta}_1, h\})$.

For all the messages in chain $y = 2$, we have the same metadata as before, but we no longer need to send $h$. Starting from $y = 3$, we need to add the number $N_{y-2}$ of messages sent in the previous sending chain (*i.e.*, chain $y-2$). We depict the key-schedule of SAID in Fig. 7.

**Comparing security models.** Our framework follows closely the model by Blazy *et al.* [7], which describes a real-or-random key-indistinguishability experiment for identity-based secure messaging. Their adversaries are either passive or active outsiders in our taxonomy. The model of [7] has several features identical to ours: a global setup, malicious-user registration procedures, sending, and receiving oracles. Since new-session instantiation is not interactive for SAID, our model boils down to Blazy *et al.*'s on this account.

However, [7] gives different leakage possibilities to its adversaries than we do, through three specific oracles (presented in the lower half of Fig. 8). The first is corruption, which yields our cross-session keys, but also all the master secret values of all ongoing sessions. $\mathcal{A}$ can also reveal a subset of cross- and single-stage keys (specified by name); by contrast, our framework only splits access by key-type (*e.g.*, querying $\mathsf{oReveal.XStage}$ yields all cross-stage keys together). Finally, [7] allows $\mathcal{A}$ black-box access to a long-term value: we denote this in Fig. 8 by a "BB" annotation. We describe in detail our classification of keys as stage-local, cross-stage, and cross-session in Table 1 and in the following paragraph.

Although $\mathsf{oReveal}$ provides $\mathcal{A}$ more fine-grained access to the local and cross-stage keys (as it can reveal them

| | $r$ | mk | ck | tag | ms | rk | rchk | ik | ID.sk | IBS.sk |
|---|---|---|---|---|---|---|---|---|---|---|
| oReveal.1Stage | ✓ | ✓ | ✓ | ✓ | | | | | | |
| oReveal.XStage | | | | | ✓ | ✓ | ✓ | | | |
| oReveal.XSid | | | | | | | | ✓ | ✓ | ✓ |
| oCorrupt | | | | | ✓ | | | ✓ | ✓ | ✓ |
| oReveal | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | |
| oHSM | | | | | BB | | | BB | BB | BB |

Figure 8: Comparison of leakage oracles for SAID (our framework and [7])

individually), the SAID protocol proofs make no use of this particular granularity: in other words, security relies on the fact that the adversary is never given access to the master secret (obtained in [7] by a oCorrupt query) simultaneously with the chain or root key allowing $\mathcal{A}$ to compute a target message key.

**The PCS-security of SAID.** As shown in Table 1, we classify the private-key material of SAID as follows:

- **Stage-specific keys:** These include, per stage: the chain and message keys at that stage, its tag, and the randomness $r$ used only once, at the beginning of the protocol;
- **Cross-stage keys:** Apart from the root and ratchet keys, cross-stage keys now include the master secret, which is input at every stage of the protocol;
- **Cross-session keys:** These include: initial private key $\mathsf{ik}$, identity-based signature $\mathsf{IBS.sk}$ and identity keys $\mathsf{ID.sk}$.

We state the following theorem, for which we provide the constructive proofs (PCS security) in Appendix C.2 and concrete attacks (PCS metric) below.

**Theorem 2** *Consider the SAID protocol modelled as a* $\mathsf{SCEKE}$ *scheme. The following results hold in the random oracle model (by replacing the KDFs and hash functions with random oracles), under the Bilinear Computational Diffie-Hellman assumption, and assuming the EUF-CMA security of the IB-signature scheme* $\mathsf{IBSig}$ *and the AKE security of the channels established between honest users and an honest* $\hat{S}$ *at registration:*

- *SAID is* $(1,0)$*-PCS secure against local outsiders (passive and active);*
- *SAID is* $(\infty,1)$*-PCS secure against local passive insiders;*
- *SAID is* $(\infty,2)$*-PCS secure against: medium passive adversaries (outsiders and insiders), and global passive attackers (outsiders and insiders);*
- *For other adversary types, SAID is* $(\infty,\infty)$*-PCS secure.*

LOCAL OUTSIDERS For both passive and active outsiders, the SAID PCS bound is $(1,0)$-PCS-secure, which is actually optimal in our framework. The main reason this holds is that the master secret (a cross-stage value) is required for each evolution; hence, an attacker can only at most learn the current message key, but no other.

LOCAL PASSIVE INSIDER Note that, according to our game, insiders might *know* a long-term secret for a user, but they will not *use them*. With its revelation oracle, $\mathcal{A}$ will be able to learn the master secret; however, since $\mathcal{A}$ cannot learn root, nor ratcheting keys, it cannot ratchet past a single message-chain.

OTHER PASSIVE ADVERSARIES For global and medium passive insider and outsider adversaries the PCS security limitations are given by the fact that $\mathcal{A}$ *can* learn a ratchet key $\mathsf{rchk}^{x,y}$ and the master secret, but, on the other hand, it is a passive attacker and can thus not *use* that ratchet key for longer than two chains. Moreover, in this case, even *passive* knowledge of $\hat{S}.\mathsf{sk}$ is not helpful. In this case, SAID is $(\infty,2)$*-PCS secure*.

OTHER ACTIVE OUTSIDERS Knowledge of the master secret is fundamental in SAID. Given this information, an active attacker can hijack the session by including fresh asymmetric ratcheting elements once the corruption has been done. Hence, as in Signal, the protocol never heals ($(\infty,\infty)$-PCS security).

ACTIVE INSIDERS We recall that the master secret keys used by the KDC at setup will now be part of the adversary's knowledge, as well as the database of entries containing identities and private keys. This allows the adversary to learn the private keys, both for signatures and their identity keys. This enables the the active, malicious KDC to impersonate Alice towards Bob and Bob towards Alice, thus endangering all their future keys ($(\infty,\infty)$-PCS security).

## 3.3 5G AKE Procedures as A SCEKE Protocol

We showcase the flexibility of our $\mathsf{SCEKE}$ framework by modelling a suite of secure-channel establishment protocols in 5th Generation Mobile Networks (5G) as a single $\mathsf{SCEKE}$ scheme. The latter will include the well-known Authentication and Key Agreement (AKA) protocol executed during Registration procedure (Reg), and a series of procedures called *handovers* [2, 3]. Unlike the two-party SAID and Signal protocols, the suite of 5G protocols we target are run between many entities – we are, nevertheless, able to model them as a two-party $\mathsf{SCEKE}$, with horizontal and vertical key-evolutions.

We briefly describe the 5G context and 5G handovers (a longer description is in Appendix D). Then we model these procedures as a $\mathsf{SCEKE}$ scheme and analyze their PCS security.

**The 5G handover protocols.** In 5G networks, mobile users (*User Equipment (UE)*) can subscribe to – and receive service from – an operator, whose back-end infrastructure is called the *core network*. The subscribers and the core will share a number of long-term cryptographic secrets denoted succinctly as $\mathsf{K}$. At any point, a user can be given mobile service through a radio "base-station" denoted Next Generation NodeB (gNB), which communicates in parallel with the UE's core network. We assume existence of mutually-authenticated gNB-to-gNB and gNB-to-core secure channels, and focus on the security of UE-to-gNB and UE-to-Core channels.

Actual mobile/application messages, called *access-stratum messages*, transit between the UEs and a gNB. During initial Registration phase (Reg), these two entities establish initial key material (via AKA protocol). Access-stratum messages are secured with an *access-stratum key* ($\mathsf{K}_{AS}$), which is derived

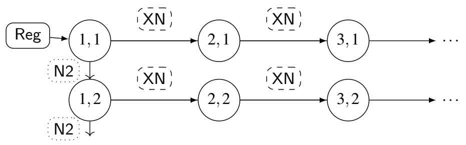


Figure 9: General model of 5G handover procedures.

from an intermediate key called $k_{gNB}$ (Signal's chain keys). The latter can be computed through a re-use of the registration procedure, or they can be obtained through evolution via handovers, as we explain next.

Handovers are required when the user connects to a new gNB (*e.g.*, because it physically moves out of reach). At this point, a *handover* is initiated, permitting the evolution of $k_{gNB}$, from the *source node (s-gNB)* shared by the UE and its old gNB, to a *tgNB*, shared by the UE and its new serving node. There are two handover procedures in 5G: XN handover procedure (XN) denoted here $\mathsf{XN}$, and N2 handover procedure (N2) denoted $\mathsf{N2}$.

**The handover procedures.** In most cases of handovers, the XN protocol is run. The s-gNB unilaterally evolves its key $k_{gNB}$ into a new key $k_{gNB}$, which s-gNB securely sends to the target target node (t-gNB). The UE will make its key $k_{gNB}$ evolve by receiving metadata from (the core and) the s-gNB over their secure channel. We will call this *horizontal evolution*.

In N2, the *core network* computes the new $k_{gNB}$ for t-gNB by refreshing a larger part of the key-schedule: a vertical evolution. The highest-level key in the 5G key-schedule that can be refreshed by N2 is denoted $k_{AMF}$. The $k_{gNB}$ keys are lower than $k_{AMF}$.

**The *5G-SCEKE* protocol.** We consider a $\mathsf{SCEKE}$ protocol that is the composition of the AKA/Reg protocol, which provides some initial key material to the user and network, and multiple, sequential runs of various handover procedures which make that key material evolve. The resulting protocol is denoted *5G-SCEKE*. Each stage corresponds to the protocol establishing, then using, a new $k_{gNB}$.

Our framework only supports two-party protocols. We thus *compress* the set of all gNB nodes and the core network into a single entity, representing the responder, Bob[7]. The initiator is Alice (the *UE*). The super-user is a key-escrow entity, associating initial key-material to sessions (abstracting AKA).

Fig. 9 presents the key evolution in *5G-SCEKE*, as a $\mathsf{SCEKE}$ protocol. Following the registration phase, Alice can horizontally evolve keys by using the $\mathsf{XN}$ procedure. When Bob wishes to respond, it runs the procedure $\mathsf{N2}$ to evolve the stage vertically. Thus, in the *5G-SCEKE* protocol the roles are asymmetric: only Alice evolves stage-keys horizontally, and only Bob evolves them vertically.

*5G-SCEKE* instantiates the initial steps of $\mathsf{SCEKE}$ with:

- **Setup:** The super-user chooses system parameters and generates $\hat{S}.\mathsf{sk}, \hat{S}.\mathsf{pk}$ for secure-channel establishment;
- **Key generation:** We assume that parties create some artificial keys $\mathsf{ik}, \mathsf{ipk}$ (non-existent in the true 5G context, but needed here in order to abstract the complexity of AKA);
- **User registration:** During user registration, each party $P$ establishes a mutually-authenticated secure channel with $\hat{S}$ and sends a registration request. Then $\hat{S}$ generates one secret $\mathsf{K}_{PQ}$ for each $Q$ in its database, but does not send them to $P$. It then updates its database with an entry indexed $P$, with tuples $(Q, K_{PQ})$ for each user $Q$ already existing.

[7]This works in our framework because we require that the endpoints to the target session to be honest – if corrupted. However, note that in some cases in the real world, parts of "Bob" might be malicious.

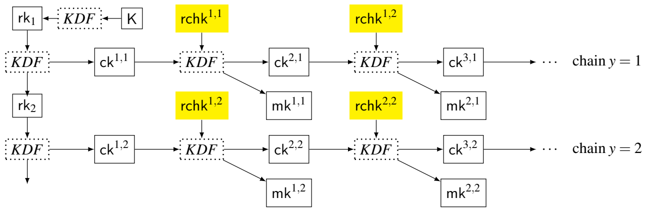


Figure 10: Generic key management for *5G-SCEKE*. The values in yellow are modifications only for *5G-SCEKE*$^{+}$.

**Instance Initialisation.** Our session instances span the entire duration of *5G-SCEKE*. When Alice initiates a session with Bob, she requests the key $\mathsf{K}A, B$ from $\hat{S}$ over a new secure channel, then uses $\mathsf{K}A, B$ as a master secret and derives, via a KDF, a root key (in practice, $\mathsf{K}_{AMF}$). A KDF computation later, the endpoints obtain the first *chain key* $\mathsf{ck}^{1,1}$ (namely $\mathsf{K}_{gNB}$) and a new root key $\mathsf{rk}_2$. The latter yields a new chain key $\mathsf{ck}^{2,1}$ and a *message key* $\mathsf{mk}^{1,1}$ (notably $\mathsf{K}_{AS}$).

Any of Alice's messages, carrying her identity as metadata, will be sufficient for Bob to initialise a session with her.

**Sending and receiving messages.** Messages are sent securely in stages, encrypted with the message keys which evolve.

- **Horizontal evolution:** When the initiator wants to send a new message ($s$ goes from $(x, y)$ to $(x+1, y)$), the chain key $\mathsf{ck}^{x+1,y}$ is fed into a KDF to get $\mathsf{ck}^{x+2,y}$ and $\mathsf{mk}^{x+1,y}$.
- **Vertical evolution:** When the responder sends a message ($s$ goes from $(x, y)$ to $(1, y+1)$), a new $k_{AMF}$ is generated from $\mathsf{rk}_2$ and fed into a KDF in order to derive $\mathsf{ck}^{1,y+1}$.

Note that the way to model *5G-SCEKE* as a $\mathsf{SCEKE}$ protocol is not unique, and as such, different results could be obtained for different variations of the protocol. Indeed, this is an advantage of our framework, as it allows us to compare those different approaches towards modelling 5G handovers.

**The PCS-security of *5G-SCEKE*.** We divide key material input into the key-schedule of *5G-SCEKE* viewed as `SCEKE` protocol:

- **Stage-specific keys:** These include chain keys, associated to $\mathsf{K}'_{gNB}$ and message keys, corresponding to $\mathsf{K}_{AS}$.
- **Cross-stage keys:** The root key, corresponding to $\mathsf{K}_{AMF}$, computed at the beginning of each chain and stored for next vertical evolution.
- **Cross-session keys:** The pre-computed key $\mathsf{K}$ shared between each two parties. Each registration procedure

corresponds to new instances of all the aforesaid keys, where K is used again.

As we describe in the Appendix (and as can be seen from Fig. 4), the *5G-SCEKE* protocol only provides healing with respect to local outsiders as it lacks any kind of unpredictable freshness. This idea lies at the core of the following improvement that we propose to *5G-SCEKE*.

***5G-SCEKE*$^+$: Our Improved-PCS *5G-SCEKE*.** We propose a simple, yet effective, modification of *5G-SCEKE* to enhance the latter's PCS-security, and denote the resulting protocol by 5G-SCEKE$^+$. Notably, we will add freshness into each horizontal evolution, thus limiting the attacker's power. These added values can be viewed boxed in yellow in Fig. 10.

Concretely, we change XN into $XN^+$, a scheme in which s-gNB does not compute the $k_{gNB}$ key for t-gNB. Instead, the latter contributes a locally-generated private value called rchk to $k_{gNB}$ (see the yellow boxes on Fig. 10). Then, t-gNB sends rchk over its secure channel to the core, which in turn forwards it to UE, encrypted with $k_{SEAF}$ (i.e., the key on top of the $k_{AMF}$ in the key-hierarchy in 5G [2, 3]). Now, the UE can also compute the new $k_{gNB}$. As the sending of rchk can be done on existing XN messages, our modification is minimal. Modifications from XN to $\mathtt{XN}^+$ are depicted in Fig. 15.

The key-material in *5G-SCEKE*$^+$ is the same as for *5G-SCEKE* except that we add the single-stage keys $\mathsf{rchk}^{x,y}$.

**The analysis of *5G-SCEKE*$^+$.** The following theorem holds for the *5G-SCEKE*$^+$ protocol.

**Theorem 3** *Consider the* 5G-SCEKE$^+$ *protocol as presented above. The following results hold in the random oracle model (by replacing the KDFs with random oracles)*

- 5G-SCEKE$^+$ *is* $(1,0)$*-PCS secure against local active outsiders and passive outsiders;*
- *For all other adversary types,* 5G-SCEKE$^+$ *is* $(\infty,\infty)$*-PCS secure.*

Formal proofs are given in Appendix E.

**Interpreting our *5G-SCEKE*$^+$ results.** Unlike Signal and SAID for which the rules of evolution are fixed and immutable, 5G handovers can be used in many different *configurations*, and the way users move within the 2D-grid of signal-providing "towers" (i.e., gNBs) impacts their healing interval. Recall that two protocols are used for evolution: XN (or $\mathtt{XN}^+$) – providing horizontal evolution, or N2 – providing vertical evolution. For instance, suppose a gNB which we denote as "Tower1" is configured to only use XN, while some "Tower2" uses only N2. Finally, some "Tower3" can be configured to use the two according to some algorithm: *e.g.*, first time XN, and then N2. If Alice comes across Tower1, then Tower3, then Tower2, she will horizontally evolve twice, then vertically once, while if she goes via Tower1, then Tower2, then Tower3, she will evolve: first horizontally, then vertically, then horizontally again.

The takeaway in this case is two-fold: first, while our results are generic, they can translate to different compromise windows for different configurations and topologies; second, configuring gNBs to use the N2 protocol often and the XN protocol seldomly is the best way of improving the actual healing provided via 5G handovers.

## 4 Discussion and Conclusion

This paper presents a framework for comparing the post-compromise security achieved by secure-channel establishment protocols featuring key-evolution. Our taxonomy of adversaries includes known adversaries in the literature, but also imagines other type of attacks. The goal of our security definition is not only to prove that key-evolution provides healing, but also to quantify how fast protocols heal. We showcase our framework by applying it to the Signal, SAID, and a composition of AKA and 5G handover protocols. Finally, we also propose a small modification to the latter protocol, which radically improves its healing speed.

Our results (see Fig. 4) indicate that optimal security (*i.e.*, (1,0)-PCS security) is achieved by SAID against local passive outsiders, as well as our improvement of 5G handovers, namely *5G-SCEKE*$^+$, against all passive outsiders. An interesting takeaway is the benefit, in *5G-SCEKE*$^+$, of using fresh, stage-specific, shared private randomness in the key-derivation process, the unpredictability of which allows us to gain stronger security than SAID for medium and global passive adversaries. However, this security comes at the expense of using shared randomness, which requires secondary secure channels.

We also indicate the benefits of the persistent authentication used in SAID to combat active session-hijacking attacks. Although the use of identifying information into the key computation can be privacy-intrusive (especially if signatures are used), it is able to provide eventual (and even speedy) healing against powerful attackers, otherwise capable of rendering a secure channel unhealable.

Through their reliance on both long-term keys and fresh asymmetric ratchets, Signal and SAID obtain better security against passive insiders than *5G-SCEKE*$^+$.

Finally, note that although active insider security is difficult to attain, it is a worthwhile goal. A takeaway of our work is that it is difficult, but essential to design protocols in which users are able to bypass the ability of superusers to create unobservable PitM attacks (for instance, one could consider two-factor authentication of the communication partner).

Our results, while insightful and strong, come with some disadvantages. We only model two-party protocols, and thus cannot analyze multi-user messaging like ART or MLS; yet as we discuss in the introduction, our framework can be applied beyond the protocols we consider, such as OTR and Wire. Moreover our approach when modelling 5G handover protocols could be applied to ratcheted key-exchange or even TLS 1.3 session resumption. We leave the quantification and comparison of such – and other – protocols as future work.

## References

[1] An open network for secure, decentralized communication, 2019. https://matrix.org/.

[2] 3GPP. System architecture for the 5G System (5GS). Technical Specification (TS) 23.501, 3rd Generation Partnership Project (3GPP), 10 2020. Version 16.0.0.

[3] 3GPP. Procedures for the 5g system. Technical Specification (TS) 23.502, 3rd Generation Partnership Project (3GPP), 10 2021. Version 16.7.0.

[4] Joël Alwen, Sandro Coretti, and Yevgeniy 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*, pages 129–158, 2019.

[5] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-14, Internet Engineering Task Force, May 2022.

[6] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, pages 232–249, 1993.

[7] Olivier Blazy, Angèle Bossuat, Xavier Bultel, Pierre-Alain Fouque, Cristina Onete, and Elena Pagnin. SAID: Reshaping Signal into an Identity-Based Asynchronous Messaging Protocol with Authenticated Ratcheting. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, pages 294–309, Stockholm, Sweden, 2019. IEEE.

[8] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, pages 213–229. Springer, 2001.

[9] Nikita Borisov, Ian Goldberg, and Eric A. Brewer. Off-the-record communication, or, why not to use PGP. In Vijay Atluri, Paul F. Syverson, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES 2004, Washington, DC, USA, October 28, 2004*, pages 77–84. ACM, 2004.

[10] Chris Brzuska, Eric Cornelissen, and Konrad Kohbrok. Cryptographic security of the MLS rfc, draft 11. *IACR Cryptol. ePrint Arch.*, page 137, 2021.

[11] Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. State separation for code-based game-playing proofs. In *Advances in Cryptology - ASIACRYPT Proceedings, Part III*, volume 11274 of *LNCS*, pages 222–249. Springer, 2018.

[12] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in cryptology – EUROCRYPT*, volume 2045 of *LNCS*, pages 453–474, 2001.

[13] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A Formal Security Analysis of the Signal Messaging Protocol. *Proceedings - 2nd IEEE European Symposium on Security and Privacy, EuroS and P 2017*, (July):451–466, 2017.

[14] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *Proceedings of ACM CCS*, page 1802–1819, 2018.

[15] Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. On post-compromise security. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 164–178, 2016.

[16] Benjamin Dowling and Britta Hale. Secure messaging authentication against active man-in-the-middle attacks. In *Proceedings of IEEE EuroS&P*, pages 54–70. IEEE, 2021.

[17] Jannik Dreier, Pascal Lafourcade, and Yassine Lakhnech. A formal taxonomy of privacy in voting protocols. In *2012 IEEE International Conference on Communications (ICC)*, pages 6710–6715, 2012.

[18] Xia Fei, Yanqin Zhu, and Xizhao Luo. Efficient identity-based signature scheme in the standard model. In *Proceedings of (ICACTE)*, volume 5, pages V5–480–V5–483, 2010.

[19] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of google's quic protocol. In *Proceedings of CCS*, page 1193–1204. ACM, 2014.

[20] R. Focardi and R. Gorrieri. A taxonomy of trace-based security properties for ccs. In *Proceedings The Computer Security Foundations Workshop VII*, pages 126–136. IEEE Computer Society, 1994.

[21] Wire Swiss GmbH. Wire security whitepaper, 2021. https://wire-docs.wire.com/download/Wire+Security+Whitepaper.pdf.

[22] M. Marlinspike and T. Perrin. The double ratchet algorithm, 2016.

[23] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. 34, 01 2010.

## A The Signal protocol

The Signal protocol can be described in terms of four operations, some only occurring once per user (like registration), some, once per session (like session setup), and others recurring at specific intervals. We depict all the steps apart from registration in Fig. 11, abstracting the way in which credentials are stored and recovered from the semi-trusted server.

- **Registration:** Each party $P$ registers by uploading on a semi-trusted server a number of (public) keys: a long-term key denoted $\mathsf{ipk}_P$, a medium-term key $\mathsf{prepk}_P$ signed with $\mathsf{ik}_P$, and optional ephemeral *public* keys $\mathsf{ephpk}_P$.

- **Session Setup:** Alice wants to initiate communication with Bob. She retrieves Bob's credentials from the server, generates an initial ratchet key-pair $(\mathsf{rchk}_A^1, \mathsf{Rchpk}_A^1)$ and an ephemeral key-pair $(\mathsf{Epk}_A, \mathsf{ek}_A)$, and uses the X3DH protocol [22] to generate an initial shared secret $\mathsf{ms}$ (master secret): $\mathsf{ms} := (\mathsf{prepk}_B)^{\mathsf{ik}_A} || (\mathsf{ipk}_B)^{\mathsf{ek}_A} || (\mathsf{prepk}_B)^{\mathsf{ek}_A} || (\mathsf{ephpk}_B)^{\mathsf{ek}_A}$. This value is used in input to a key derivation function ($\mathsf{KDF}_r$), outputting the root key $\mathsf{rk}_1$ and the chain key $\mathsf{ck}^{1,1}$. The latter is used to derive the first message key $\mathsf{mk}^{(1,1)}$ that Alice uses to communicate with Bob. The following associated data (AD) is appended to that message: the value 1 (for the index $x$), Alice's ephemeral public key $\mathsf{Epk}_A$, the ratchet key $\mathsf{Rchpk}^1$, as well as Alice's and Bob's identities.

- **Symmetric Ratchet:** Whenever a sender $P$ chooses a new message to send, the stage changes from $(x,y)$ to $(x+1,y)$ and a new symmetric ratchet takes place. At stage $(x,y)$, the message key is $\mathsf{mk}^{x,y}$, derived from a stage secret $\mathsf{ck}^{x,y}$. In fact, given $\mathsf{ck}^{x,y}$, the sender computed (at stage $(x-1,y)$) the values $\mathsf{ck}^{x+1,y}$ and $\mathsf{mk}^{x,y}$. At stage $(x+1,y)$, the sender inputs $\mathsf{ck}^{x+1,y}$ to the key-derivation function $\mathsf{KDF}_m$ and receives the output $\mathsf{ck}^{x+2,y}$ and $\mathsf{mk}^{x+1,y}$. The key $\mathsf{mk}^{x+1,y}$ is then used for the authenticated encryption of the sender's message at stage $(x+1,y)$. The AD sent at this stage will be the ratchet key $\mathsf{Rchpk}^y$ and the stage index[8] $x+1$. The same process takes place on the receiving side, in order to authenticate and decrypt messages.

- **Asymmetric Ratchet:** If the speaker changes (that is, Alice stops sending messages and Bob starts instead, or vice-versa), the new speaker inserts fresh Diffie-Hellman elements into the key-derivation. Assume that we are at stage $(x,y)$ and the speaker changes (thus yielding stage $(1,y+1)$). Different computations are made depending on whether the new speaker is the initiator or the responder.

  1. First assume that initiator Alice was the speaker at stages $(\cdot,y)$; therefore $y$ is even at each stage $(\cdot,y)$ and the encrypted message included associated data $\mathsf{Rchpk}^y$. When Bob comes online, he chooses a new ratchet key $\mathsf{rchk}^{y+1}$, and the public key $\mathsf{Rchpk}^{y+1}$ is then computed. A temporary value $t$ and the chain key $\mathsf{ck}^{(1,y+1)}$ are calculated from the root key[9] $\mathsf{rk}_y$ and the Diffie-Hellman product $(\mathsf{Rchpk}^y)^{\mathsf{rchk}^{y+1}}$ via $\mathsf{KDF}_r$. Then, the chain and message keys are computed as described in the previous item. From that point onwards, keys evolve by symmetric ratcheting until the speaker changes again.

  2. Now assume that the responder was the speaker at stages $(\cdot,y)$; therefore $y$ is odd and at each stage $(\cdot,y)$ the encrypted message includes associated data $\mathsf{Rchpk}^y$. When Alice comes online, she chooses new ratcheting information $\mathsf{rchk}^{y+1}, \mathsf{Rchpk}^{y+1}$ and computes a new root key $\mathsf{rk}_{y+1}$ and the base chain key $\mathsf{ck}^{(0,y+1)}$ from the value $t$ computed at stage $(1,y)$ (see previous bullet point) and the Diffie-Hellman product $(\mathsf{Rchpk}^y)^{\mathsf{rchk}^{y+1}}$. From here the key derivation proceeds as described in the bullet point on symmetric ratcheting.

### A.1 The insecurity of Signal

In Signal, Alice and Bob each has a long-term private value: their identity keys $\mathsf{ik}_A$ and $\mathsf{ik}_B$ respectively. However, that key is only embedded into each session at one point: in the calculation of the master secret $\mathsf{ms}$. The protocol also uses the responder's medium-term key $\mathsf{prek}_B$ (with corresponding public key $\mathsf{prepk}_B$).

During each protocol session, the end users also compute a number of session-specific values: the master secret, ratchet keys, root keys, chain keys, and ultimately, message keys. Some of these values (like used message keys) are removed from memory after a relatively brief lifetime, while others (such as root, ratcheting, and to some extent, chain keys) are retained for longer.

Cohn-Gordon *et al.* [13] proved the security and healing properties for a modified version of Signal, in which metadata including ratchet public keys are sent – not as part of the associated data in ciphertexts – but outside of the encryption.

[8] In the original protocol, the sender also sends the identity public keys of Alice and Bob; since these values are public and constant for all stages, we omit them.

[9] Root keys are only computed when one reverts back to the initiator, so in our notation, on stages $(1,y)$ for even values of $y$.

| **Alice** $(\mathsf{ik}_A, \mathsf{ipk}_B, \mathsf{prepk}_B, \mathsf{ephpk}_B)$ | | **Bob** $(\mathsf{ik}_B, \mathsf{ipk}_A, \mathsf{prek}_B, \mathsf{ephk}_B)$ |
|---|---|---|
| | **Session initialization**: initiator Alice, responder Bob. | |
| Generate: $\mathsf{ek}_A, \mathsf{rchk}^1, \xleftarrow{\$} \mathbb{Z}_q$,<br>Compute: $\mathsf{Epk}_A \leftarrow g^{\mathsf{ek}_A}$; $\mathsf{Rchpk}^1 \leftarrow g^{\mathsf{rchk}^1}$;<br>Compute: $ms \leftarrow \mathsf{prepk}_B^{\mathsf{ik}_A} \| \mathsf{ipk}_B^{\mathsf{ek}_A} \| \mathsf{prepk}_B^{\mathsf{ek}_A} \| \mathsf{ephpk}_B{}^{\mathsf{ek}_A}$<br>Initial keys: $\mathsf{rk}_1, \mathsf{ck}^{1,1} \leftarrow \mathsf{KDF}_r(\mathsf{prepk}_B^{\mathsf{rchk}^1} \| ms)$<br>$\mathsf{ck}^{2,1}, \mathsf{mk}^{1,1} \leftarrow \mathsf{HKDF}(\mathsf{ck}^{1,1})$ | | |
| | **First message**: stage $(1,1)$, Alice is the sender, Bob, the receiver. | |
| Set $\mathsf{AD}_{1,1} \leftarrow (x=1) \| \mathsf{Rchpk}^1 \| \mathsf{Epk}_A \| \mathsf{ipk}_B \| \mathsf{ipk}_A$ | $\xrightarrow{c \leftarrow \mathtt{AE}_{\mathsf{mk}^{1,1}}(M_{1,1} \vert \mathsf{AD}=\mathsf{AD}_{1,1})}$ | Compute: $ms \leftarrow \mathsf{ipk}_A^{\mathsf{prek}_B} \| \mathsf{Epk}_A^{\mathsf{ik}_B} \| \mathsf{Epk}_A^{\mathsf{prek}_B} \| \mathsf{Epk}_A^{\mathsf{ephk}_B}$<br>Set: $\mathsf{ck}^{1,1} \leftarrow \mathsf{KDF}_r((\mathsf{Rchpk}^1)^{\mathsf{prek}_B} \| ms)$<br>and: $\mathsf{ck}^{2,1}, \mathsf{mk}^{1,1} \leftarrow \mathsf{HKDF}(\mathsf{ck}^{1,1})$<br>AE decrypt $c$ to $M_{1,1}$. |
| | $\ell$-th **message**: stage $(\ell,1)$, Alice is the sender, Bob, the receiver. | |
| Stage keys: $\mathsf{ck}^{\ell+1,1}, \mathsf{mk}^{\ell,1} \leftarrow \mathsf{HKDF}(\mathsf{ck}^{\ell,1})$<br>Set $\mathsf{AD}_{\ell,1} \leftarrow (x=\ell) \| \mathsf{Rchpk}^1 \| \mathsf{ipk}_B \| \mathsf{ipk}_A$ | $\xrightarrow{c \leftarrow \mathtt{AE}_{\mathsf{mk}^{\ell,1}}(M_{\ell,1} \vert \mathsf{AD}=\mathsf{AD}_{\ell,1})}$ | Set $\mathsf{ck}^{\ell+1,1}, \mathsf{mk}^{\ell,1} \leftarrow \mathsf{HKDF}(\mathsf{ck}^{\ell,1})$<br>AE decrypt $c$ to $M_{\ell,1}$. |
| | **Switching speakers**: Bob comes online and begins a new ratcheting chain. | |
| | | $\mathsf{rchk}^2 \xleftarrow{\$} \mathbb{Z}_q$<br>Set $tmp, \mathsf{ck}^{1,2} \leftarrow \mathsf{KDF}_r(\mathsf{rk}_1, \mathsf{Rchpk}^1{}^{\mathsf{rchk}^2})$<br>and: $\mathsf{ck}^{2,2}, \mathsf{mk}^{1,2} \leftarrow \mathsf{HKDF}(\mathsf{ck}^{1,2})$ |
| | **Bob's message, stage** $(1,2)$: Bob is the sender, Alice is the receiver. | |
| | | Set $\mathsf{AD}_{1,2} \leftarrow (x=2) \| \mathsf{Rchpk}^2 \| \mathsf{ipk}_B \| \mathsf{ipk}_A$ |
| Set $tmp, \mathsf{ck}^{1,2} \leftarrow \mathsf{KDF}_r(\mathsf{rk}_1, \mathsf{Rchpk}^2{}^{\mathsf{rchk}^1})$<br>and: $\mathsf{ck}^{2,2}, \mathsf{mk}^{1,2} \leftarrow \mathsf{HKDF}(\mathsf{ck}^{1,2})$<br>AE decrypt $c$ to $M_{\ell,1}$. | $\xleftarrow{c \leftarrow \mathtt{AE}_{\mathsf{mk}^{1,2}}(M_{1,2} \vert \mathsf{AD}=\mathsf{AD}_{1,2})}$ | |
| | **Second speaker switch**: Alice is back online. | |
| $\mathsf{rchk}^3 \xleftarrow{\$} \mathbb{Z}_q$<br>Set $\mathsf{rk}_2, \mathsf{ck}^{1,3} \leftarrow \mathsf{KDF}_r(tmp, \mathsf{Rchpk}^2{}^{\mathsf{rchk}^3})$<br>and: $\mathsf{ck}^{2,3}, \mathsf{mk}^{1,3} \leftarrow \mathsf{HKDF}(\mathsf{ck}^{1,3})$ | | |

Figure 11: A Signal protocol session between initiator Alice and responder Bob.

The security statement takes into account the delay in healing (*i.e.*, up to two full message chains), and also rules out active hijacking attacks. Their results also factor in a semi-trusted (rather than potentially-malicious) centralized server.

**On compromising Alice.** There are different ways in which an attacker can compromise Alice. If, for instance, an adversary obtains a message key $\mathsf{ck}^{x,y}$, then it will be able to got through the remainder of the key-derivations left in that message chain. An attacker in possession of Alice's ratchet key $\mathsf{rchk}^{1,2k+1}$ (for some integer value of $k$) can compute the next Diffie-Hellman value; with the aid of the input root key $\mathsf{rk}_k$ this will directly yield all the keys for the next chain of messages.

We depict in Fig. 12 one of the worst ways in which an adversary can compromise Alice, without actually obtaining her long-term private keys. In this case, the adversary has obtained the master secret value ms and Alice's ratchet key $\mathsf{rchk}^{0,1}$. Using these values, it is able to compromise two full message chains, even in the absence of any further active attacks.

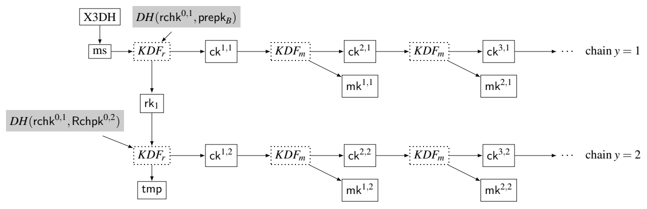


Figure 12: The key schedule of Signal where the DH values are marked with grey boxes. Each stage $(x,y)$ has its x-coordinate corresponding to a message (horizontal moves) inside a chain (vertical moves) for y-coordinate.

**Active attacks.** If an adversary actively uses the values it has compromised, it can hijack a session. This is because in Signal, parties are authenticated by their knowledge of past keys, rather than by long-term values. Moreover, because Diffie-Hellman keys are symmetric in the way they are computed, the attacker can actually impersonate Alice to Bob, or Bob to Alice (or both).

Say that we are in the same situation as in Fig. 12. The adversary has learned ms and the ratchet private key $\mathsf{rchk}^{0,1}$.

The attacker's next goal is to hijack the session from Alice, pretending to be her in her conversation with Bob. To do so, the adversary can suppress all the messages coming from the genuine Alice (or, in fact, transmit them, or transmit modifications of those messages: after all, the attacker has all the keys). When Bob ratchets and sends his first messages, our adversary can continue to read those, because knowing $\mathsf{rchk}^{0,1}$ allows it to compromise the security of all the messages sent by Bob. Then, instead of allowing the genuine Alice to insert the Diffie-Hellman share which would actually heal the protocol from the compromise, the adversary chooses a ratchet key $\mathsf{rchk}^{0,3}$ and mirrors the steps Alice should have taken to ratchet in message-chain $y = 3$. From this point:

- Bob falsely believes it is still talking to Alice, across a channel that is secure with respect to a Person-in-the-Middle adversary;
- In fact, Bob is talking to the adversary over a channel that is secure with respect to all outsiders, including the genuine Alice.

A more insidious attacker would not stop there. At this point, Alice is locked out of her channel with Bob: in realising this, she might raise the alarm, thus allowing Bob to realize the hijack.

However, in Signal, compromising Alice's state also allows the attacker to hijack the session from Bob, thus impersonating the latter to Alice. Instead of allowing Bob to ratchet, the adversary picks its own Diffie-Hellman element $\mathsf{rchk}^{0,2}$ and using its knowledge of the master secret value as a shortcut, performs the steps included under the heading "Switching speakers" in Fig. 11. From this point onwards:

- Alice will believe it is still talking to Bob, across a channel that is secure with respect to a Person-in-the-Middle adversary;
- In fact, Alice will be talking to the adversary over a channel that is secure with respect to all outsiders, including the genuine Bob.

The adversary can continue to play its Person-in-the-Middle part, and can choose to either benignly forward the conversation between Alice and Bob, or insert its own messages into it. In the former case, the security of the channel will be breached forever. In the latter, the adversary can choose the information it will forward to Alice and Bob, potentially extracting from them some highly-sensitive information, or feeding them malicious input.

Most importantly, the adversary can achieve this by just learning some of Alice's session-specific information. It need not even be the master secret and corresponding ratcheting key: the adversary could instead corrupt the root key and ratchet key for some message chain where Alice is the sender, or the temporary value *tmp* and Bob's ratchet key for a message chain where Bob is the sender. Unlike the master secret, such values need to be kept in the device's memory for a longer time, so as to allow the party to ratchet.

**The credential server.** In Signal, users register (public-)key bundles, which are uploaded onto a centralized server. These bundles include identity keys $\mathsf{ipk}_P$, medium-term pre-keys $\mathsf{prepk}_P$, ephemeral keys $\mathsf{ephpk}_P$, and a signature keyed with $\mathsf{ipk}_P$ over $\mathsf{prepk}_P$.

In order to start a session, both the initiator and the responder request their partner's credentials from the server – which is trusted to forward that data faithfully. If, on the contrary, the server misbehaves, it can insert Person-in-the-Middle adversaries in every single new session. The strategy will be the same every time:

- Instead of forwarding to Alice the credentials of its desired communications' partner (say Bob), the server will forward the credentials of a Person-in-the-Middle Charlie – either colluding with the server or registered by the server itself. From this point onwards, Alice will communicate with Charlie, believing the latter is Bob;
- Simultaneously, Charlie will recover Bob's credentials from the centralized server (which this time behaves honestly), and start a session with the latter, claiming to be Alice (though including Charlie's own identity key);
- If queried by Bob for Alice's credentials, the server will respond with Charlie's data. From this point on, Bob will believe it is communicating with Alice, but instead will be talking to Bob.
- Subsequently, Charlie can choose to either forward messages between Alice and Bob, or inject its own messages in the conversation.

## B The SAID protocol

The SAID protocol was designed in the identity-based paradigm, in which parties can derive the public key of their interlocutors from their identities. The corresponding private keys are picked securely by a special entity (the Key Distribution Center), which first runs a global setup, then generates and distributes the private keys to protocol participants.

In this section, we use an identity-based signature scheme which supersedes Signal's key storage server, thus providing persistent authentication and the additional advantage of public verification with respect to a known identity (in lieu of a given verification key).

**Identity-based signatures.** An identity-based signature [18] scheme is made up of four possibly randomized algorithms $\mathsf{IBSig}$=($\mathtt{aIBS.Setup}$, $\mathtt{aIBS.Extract}$, $\mathtt{aIBS.Sign}$, $\mathtt{aIBS.Vfy}$) with the following properties:

- aIBS.Setup: takes in input a security parameter $n$ (in unary) and outputs: public parameters IBS.ppar, a public key IBS.mpk, and a private key IBS.msk;

- aIBS.Extract: takes in input all public parameters, the master secret key msk, and an identity of a user $P$, and outputs a private signing key for that user: $\mathsf{IBS.sk}_P$;

- aIBS.Sign: takes in input all public parameters, a private key $\mathsf{IBS.sk}_P$, and a message $M$, and outputs a signature $\sigma$;

- aIBS.Vfy: takes in input the public parameters, the identity $P$ of the purported signer, a message $M$ and a signature $\sigma$, and outputs a bit, either 1 if the signature verifies for the given identity and message, and 0 otherwise.

The Existential Unforgeability against Chosen-Message Attacks (EUF-CMA) notion for identity-based signatures is similar to the one for tradition signature schemes, and we do not recall it here, referring users to [18] instead.

## B.1 Protocol description

We proceed to describe how users Alice and Bob (denoted $A$ and $B$), with Alice playing the role of the initiator, can register, start, and run a session of our protocol. In the interest of clarity, we will use the same notation for this protocol as for the presentation of Signal. Specifically: we begin counting stages at 1 on both the horizontal and vertical components; we use the horizontal component (x) to index messages from the same sender, while the vertical one is used for switching speakers; our notations for ratchet keys only contain the stage (as this immediately implies their owner); and we use chain keys $\mathsf{ck}^{x,y}$ instead of base keys, and message keys $\mathsf{mk}^{x,y}$ instead of keys.

The SAID protocol has four main phases:

- **Parameter Generation:** run once, by a trusted party (typically the KDC), to set up the public parameters of the protocol;

- **User Registration** allow users to register to the KDC, thus receiving their identity-based cryptographic data;

- **Session Initialization** performed by a user $A$ to begin a chat with a registered user $B$. In this phase, $A$ generates a long-term master-secret which will then be used throughout the protocol (entering as input to the aSend algorithm);

- **Messaging** takes place when two users communicate in a session. This phase in characterized by sequences of symmetric and asymmetric ratchets.

**Parameter generation.** The Key-Distribution Center (KDC) will first set up the mathematical structure within which the SAID protocol will run. These parameters are universal to all the users and all the sessions that will ever be run.

- For the identity-based signature scheme, we need to generate public parameters IBS.ppar and a master key-pair (IBS.msk, IBS.mpk);

- For the AEAD cipher suites, the KDC will need to establish the set of possible keys, nonces, messages, and headers;

- We generate groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ generated by $g_1, g_2$, and $g_T$ respectively, and a bilinear pairing $e$

- In addition to the IB Signature scheme, we will also need to embed user identities into group keys. To do so, we need a master secret key and master public key: $\mathsf{ID.msk} \xleftarrow{\$} \mathbb{Z}_p$ and $\mathsf{ID.mpk} = g_2^{\mathsf{ID.msk}} \in \mathbb{G}_2$;

- The KDC chooses hash functions $H, H_2$ with range $\mathbb{G}_1^*$;

- The protocol requires two key-derivation functions (KDFs), like in the case of Signal: $\mathsf{KDF}_r$ for root- and chain-key derivation, and $\mathsf{KDF}_m$ for message and chain-key derivation (typically instantiated as HKDF).

The cumulative public parameters pparam output by the KDC will include all the *public* values and algorithms above. The master secrets, kept by KDC only, are IBS.msk and ID.msk.

**User Registration.** A user $A$ registers to the system by sending her identity, $A$, to the KDC. The KDC returns the user's secret signing key $\mathsf{IBS.sk}_A \leftarrow \mathtt{aIBS.Extract}(\mathsf{IBS.ppar}, \mathsf{IBS.msk}, A)$ and her secret identification key $\mathsf{ID.sk}_A \in \mathbb{G}_1$ generated as[10] $\mathsf{ID.sk}_A = H_2(A)^{\mathsf{ID.msk}}$. The KDC also adds $A$ into a list of registered users, and replies to any future attempt to register $A$ with the error message 'username taken'.

**Session Initialization.** In SAID, any registered user $A$ can initiate a session with another registered user $B$ (without requiring the online presence of the KDC), following the procedure depicted at the beginning of Figure 13, which we also detail below.

$A$ begins by choosing a random ratchet secret key $\mathsf{rchk}^1 \xleftarrow{\$} \mathbb{Z}_p$ and computes its corresponding public key $\mathsf{Rchpk}^1 = g_1^{\mathsf{rchk}^1}$. As in Signal, these ratchet keys are not used yet, but the target responder $B$ will need them to make his first asymmetric ratchet and respond to $A$'s messages. In addition, $A$ picks a random value $r \xleftarrow{\$} \mathbb{Z}_p$ and computes $h = g_2^r$. At this point $A$ uses its identity-based signature credentials to generate a signature on the metadata of the first message chain $\mathsf{meta}_1 \leftarrow (A, B, \mathsf{Rchpk}^1)$ and the public value $h$:

$$\sigma \leftarrow \mathtt{aIBS.Sign}(\mathsf{IBS.ppar}, \mathsf{IBS.sk}_A, \{\mathsf{meta}_1, h\}).$$

The values $h$ and $\sigma$ will be part of the associated data (AD) of all the messages sent by Alice along the first

[10]The user's secret identification key is essentially a Boneh-Franklin key for identity based encryption [8].

**Alice** $(A, \mathsf{IBS.sk}_A, \mathsf{ID.sk}_A)$ **Bob** $(B, \mathsf{IBS.sk}_B, \mathsf{ID.sk}_B)$

**Session initialization**: initiator Alice, responder Bob.

Generate: $\mathsf{rchk}^1, r, \mathsf{tag}^{1,1} \xleftarrow{\$} \mathbb{Z}_p$
Compute: $\mathsf{Rchpk}^1 \leftarrow {g_1}^{\mathsf{rchk}^1}$ and $h \leftarrow g_2^r$
Let: $\mathsf{meta}_1 \leftarrow (A, B, \mathsf{Rchpk}^1)$
$\sigma \leftarrow \mathtt{aIBS.Sign}(\mathsf{IBS.ppar}, \mathsf{IBS.sk}_A, \{\mathsf{meta}_1, h\})$
Compute: $\mathsf{ms}_{AB} \leftarrow e(H(B), \mathsf{ID.mpk}^r)$
Initial keys: $(\mathsf{rk}_1, \mathsf{ck}^{1,1}) \leftarrow \mathsf{KDF}_r(\mathsf{ms}_{AB}, g_1)$
$(\mathsf{mk}^{1,1}, \mathsf{ck}^{2,1}) \leftarrow \mathsf{KDF}_m(\mathsf{ms}_{AB}, \mathsf{ck}^{1,1}, \mathsf{tag}^{1,1})$

**First message**: stage $(1,1)$, Alice is the sender, Bob, the receiver.

$\mathsf{AD}_{1,1} \leftarrow (\mathsf{meta}_1, h, x = 1, \mathsf{tag}^{1,1}, \sigma)$

$\xrightarrow{c \leftarrow \mathtt{AE}_{\mathsf{mk}^{1,1}}(M_{1,1} | \mathsf{AD} = \mathsf{AD}_{1,1})}$

Let: $\mathsf{meta}_1 \leftarrow (A, B, \mathsf{Rchpk}^1)$
Check $1 = \mathtt{aIBS.Vfy}(\mathsf{IBS.ppar}, A, \{\mathsf{meta}, h\}, \sigma)$
Compute $\mathsf{ms}_{AB} \leftarrow e(\mathsf{ID.sk}_B, h)$
Initial keys: $(\mathsf{rk}_1, \mathsf{ck}^{1,1}) \leftarrow \mathsf{KDF}_r(\mathsf{ms}_{AB}, g_1, \mathsf{tag}^{1,1})$
$(\mathsf{mk}^{1,1}, \mathsf{ck}^{2,1}) \leftarrow \mathsf{KDF}_m(\mathsf{ms}_{AB}, \mathsf{ck}^{1,1})$
AE decrypt $c$ to $M_{1,1}$.

**$\ell$-th message**: stage $(\ell, 1)$, Alice is the sender, Bob, the receiver.

Generate: $\mathsf{tag}^{\ell,1} \xleftarrow{\$} \mathbb{Z}_p$
Keys: $(\mathsf{mk}^{\ell,1}, \mathsf{ck}^{\ell+1,1}) \leftarrow \mathsf{KDF}_m(\mathsf{ms}_{AB}, \mathsf{ck}^{\ell,1}, \mathsf{tag}^{\ell,1})$
Set $\mathsf{AD}_{\ell,1} \leftarrow (\mathsf{meta}_1, h, (x = \ell), \mathsf{tag}^{\ell,1}, \sigma)$

$\xrightarrow{c \leftarrow \mathtt{AE}_{\mathsf{mk}^{\ell,1}}(M_{\ell,1} | \mathsf{AD} = \mathsf{AD}_{\ell,1})}$

Set $(\mathsf{mk}^{\ell,1}, \mathsf{ck}^{\ell+1,1}) \leftarrow \mathsf{KDF}_m(\mathsf{ms}_{AB}, \mathsf{ck}^{\ell,1}, \mathsf{tag}^{\ell,1})$
AE decrypt $c$ to $M_{\ell,1}$.

**Switching speakers**: Bob comes online and begins a new ratcheting chain.

Generate: $\mathsf{rchk}^2, \mathsf{tag}^{1,2} \xleftarrow{\$} \mathbb{Z}_p$
Compute: $\mathsf{Rchpk}^2 \leftarrow {g_1}^{\mathsf{rchk}^2}$
Set: $\Delta^2 \leftarrow \left(\mathsf{Rchpk}^1\right)^{\mathsf{rchk}^2}$
Compute: $(\mathsf{rk}_2, \mathsf{ck}^{1,2}) \leftarrow \mathsf{KDF}_r(\mathsf{ms}_{AB}, \Delta^2, \mathsf{rk}_1)$
$(\mathsf{mk}^{1,2}, \mathsf{ck}^{2,2}) \leftarrow \mathsf{KDF}_m(\mathsf{ms}_{AB}, \mathsf{ck}^{1,2}, \mathsf{tag}^{1,2})$

**Bob's message, stage** $(1,2)$: Bob is the sender, Alice is the receiver.

Let: $\mathsf{meta}_2 \leftarrow (A, B, \mathsf{Rchpk}^2)$
$\mathsf{AD}_{1,2} \leftarrow (\mathsf{meta}_2, (x = 1), \mathsf{tag}^{1,2})$

$\xleftarrow{c \leftarrow \mathtt{AE}_{\mathsf{mk}^{1,2}}(M_{1,2} | \mathsf{AD} = \mathsf{AD}_{1,2})}$

Set: $\Delta^2 \leftarrow \left(\mathsf{Rchpk}^2\right)^{\mathsf{rchk}^1}$
Compute: $(\mathsf{rk}_2, \mathsf{ck}^{1,2}) \leftarrow \mathsf{KDF}_r(\mathsf{ms}_{AB}, \Delta^2, \mathsf{rk}_1)$
$(\mathsf{mk}^{1,2}, \mathsf{ck}^{2,2}) \leftarrow \mathsf{KDF}_m(\mathsf{ms}_{AB}, \mathsf{ck}^{1,2}, \mathsf{tag}^{1,2})$
AE decrypt $c$ to $M_{\ell,1}$

**Second speaker switch**: Alice is back online.

Generate: $\mathsf{rchk}^3, \mathsf{tag}^{1,3} \xleftarrow{\$} \mathbb{Z}_p$
Compute: $\mathsf{Rchpk}^3 \leftarrow {g_1}^{\mathsf{rchk}^3}$
Set: $\Delta^3 \leftarrow \left(\mathsf{Rchpk}^2\right)^{\mathsf{rchk}^3}$
Compute: $(\mathsf{rk}_3, \mathsf{ck}^{1,3}) \leftarrow \mathsf{KDF}_r(\mathsf{ms}_{AB}, \Delta^3, \mathsf{rk}_2)$
$(\mathsf{mk}^{1,3}, \mathsf{ck}^{2,3}) \leftarrow \mathsf{KDF}_m(\mathsf{ms}_{AB}, \mathsf{ck}^{1,3}, \mathsf{tag}^{1,3})$
Let: $\mathsf{meta}_3 \leftarrow (A, B, \mathsf{Rchpk}^3)$
$\mathsf{AD}_{1,3} \leftarrow (\mathsf{meta}_3, (x = 1), N_1, \mathsf{tag}^{1,3})$

Figure 13: The SAID protocol. Note that, for message-chains with index higher than 2, parties add to their associated data the number of messages they had sent at the immediately-previous message chain for which they played the part of senders.

message-chain. The master secret shared between $A$ and $B$ is $\mathsf{ms}_{AB} = e(H(B), \mathsf{ID.mpk})^r$. To generate the initial root key $\mathsf{rk}_1$ and chain-key $\mathsf{ck}^{1,1}$, the values $(\mathsf{ms}_{AB}, g_1)$ are input to $\mathsf{KDF}_r$. By using the computed $\mathsf{ck}^{1,1}$, $A$ can perform its first *symmetric ratchet*, obtaining the message key $\mathsf{mk}^{1,1}$ and channel key $\mathsf{ck}^{2,1}$ as the output of $\mathsf{KDF}_m(\mathsf{ms}_{AB}, \mathsf{ck}^{1,1}, \mathsf{tag}^{1,1})$ for the freshly-generated $\mathsf{tag}^{1,1}$. Finally, $A$ authenticates and encrypts the message $M_{1,1}$ with associated data $\mathsf{AD} = (\mathsf{meta}_1, h, (1,1), \sigma)$, sending the resulting ciphertext $c$ to Bob.

Upon receiving this ciphertext, Bob first verifies the identity-based signature $\sigma$, then retraces Alice's steps to obtain the message key that will allow it to authenticate and decrypt the message it has received.

**Messaging.** Following the way Signal works, in SAID the key material also evolves through symmetric and asymmetric ratcheting.

Symmetric Ratcheting. A user performs a symmetric ratchet when she wishes to obtain a chain- and a message-key, to either encrypt one *more* message, without having received a reply; or to decrypt one more message before responding. In particular, recall that a symmetric ratchet increases the $x$ counter of the stage, so if the starting stage is $(x,y)$, after the symmetric ratchet we land at stage $(x+1,y)$.

The process of a symmetric ratchet, also show in the transition from stage $(1,1)$ to $(\ell,1)$ in Fig. 13 also shows user Alice using the shared master secret and the chain key of stage $(x,y)$ to output the message key for stage $(x,y)$ and the chain-key for stage $(x+1,y)$.

Note that, as it is in Signal, $\mathsf{KDF}_m$ is split into two parts, as shown below: one which generates the next chain-key, and one which generates the encryption key. Only the latter of these two uses the random tag as input, in order to handle out-of-order messages, *i.e.*, the chain-keys could be computed simply from the previous ones, but not the encryption keys.

$$(\mathsf{ms}_{AB}, \mathsf{ck}^{x,y}) \xrightarrow{\mathsf{HMAC}} \mathsf{ck}^{x+1,y}$$
$$(\mathsf{ms}_{AB}, \mathsf{ck}^{x,y}, \mathsf{tag}^{x,y}) \xrightarrow{\mathsf{HMAC}} t \xrightarrow{\mathsf{HKDF}} \mathsf{mk}^{x,y}$$

The random tag will be included in the associated data to enable the responder to generate the same key $\mathsf{mk}^{x,y}$.

Asymmetric Ratcheting. Whenever a message is sent by the party who is not the sender of the *last* message in the chat, an asymmetric ratchet happens. Asymmetric ratcheting increases the $y$ counter and resets the $x$ counter of the chat state, so if the starting stage is $(x,y)$, after the asymmetric ratchet we land at stage $(1,y+1)$.

As depicted in Fig. 13, assuming $A$ was the sender at stage $(x,y)$, then, to send his response $B$ selects a random ratchet secret key $\mathsf{rchk}^y$ and computes the shared secret $\Delta^y = (\mathsf{Rchpk}^{y-1})^{\mathsf{rchk}^y}$. He then inputs the shared master secret, the newly computed secret value $\Delta^y$, and the current root key (of level $y-1$) to $\mathsf{KDF}_r$ and obtains the root key for message-chain $y$, together with the new chain-key for stage $(1,y)$. Finally, $B$ performs a symmetric ratchet to generate the message-key for stage $(1,y)$ (and the next base-key for stage $(2,y)$).

Note that, furthermore, as depicted in Fig. 13, the associated data sent along with the message at stage $(x,y)$ contains Alice and Bob's identity, the ratchet public key of the current sender for message-chain $y$, the horizontal-index counter $(x=1)$, and the number $N_{y-1}$ of messages sent by the same party at level $y-2$ (this value is not used for message-chains $y=1$ and $y=2$), and, finally, the tag $\mathsf{tag}^{x,y}$.

## C Security proofs for Signal and SAID

We consider the different cases corresponding to the behaviour of the adversary. We give an overview of our proofs in Figure 14.

**Conventions.** We assume that all KDFs are modelled as random oracles. Each key $k$ has $|\mathbb{k}|$ values. Note that the key space might be of same size for all keys (*e.g.*, $|\mathbb{k}|$, the order of the group for all $k$). The security statements are parametrized by the maximal number of stages $\mathsf{n_S}$, the maximal number of message $\mathsf{n_{x-max}}$ in a given chain, the maximal number of chain $\mathsf{n_{y-max}}$, run by any given instance, the number of parties generated by the adversary $\mathsf{n_P}$ and the number of sessions $\mathsf{n_\pi}$ created by any given party. Finally, we consider all calls to $\mathsf{KDF}$ as queries to random oracles.

The proofs are organized through game hops where the first game is the original security game (see Figure 3 of Sec. 2.4).

### C.1 Games for Signal

$\mathbb{G}_0$ : This game corresponds to the original security game (Fig. 3 of Sec. 2.4). The advantage of $\mathcal{A}$ against this game is $\mathsf{Adv}_0$.

$\mathbb{G}_1$ : In this game $\mathcal{C}$ guesses $P$, $Q$, the session index of the target session, and the target stage $s^\star = (x^\star, y^\star)$ for which $\mathcal{A}$ has queried $\mathsf{oTest}$.

If $\mathcal{A}$ queries another parties, session or tested stage then $\mathcal{C}$ aborts the game and returns a random bit. Therefore we have the following:

$$\mathsf{Adv}_0 \leq \mathsf{n_P}^2 \cdot \mathsf{n_\pi} \cdot \mathsf{n_{x-max}} \cdot \mathsf{n_{y-max}} \cdot \mathsf{Adv}_1$$

The next games are dedicated to ensure that no DH values collide. Moreover, we assume the uniqueness of the identity key for each party.

$\mathbb{G}_2$ : This game is the same as $\mathbb{G}_1$ except that the challenger aborts if two values $\mathsf{ephk}$ collide. We have:

$$\mathsf{Adv}_1 \leq \binom{\mathsf{n_\pi}}{2} \cdot 2^{-|\mathsf{ephk}|} + \mathsf{Adv}_2$$

At this point, the uniqueness of the master secret $\mathsf{ms}$ is guaranteed. Indeed, $\mathsf{ms}$ is computed using $\mathsf{ik}$ and also $\mathsf{ephk}$ thus by uniqueness of the former and the latter, we have uniqueness of the shared secret $\mathsf{ms}$. Moreover, the sessions are also unique by uniqueness of the ephemeral keys.

$\mathbb{G}_3$ : We modify the previous game to avoid collisions of honestly-generated ratchet keys $\mathsf{rchk}$. We have:

$$\mathsf{Adv}_2 \leq \binom{\mathsf{n_\pi} \cdot \mathsf{n_{y-max}}}{2} \cdot 2^{-|\mathsf{rchk}|} + \mathsf{Adv}_3$$

$\mathbb{G}_4$ : In this game, we ensure that there is no collision for honestly-generated $\mathsf{prek}$. We can upper-bound the total number of pre-keys by the number of sessions $\mathsf{n_\pi}$:

$$\mathsf{Adv}_3 \leq \binom{\mathsf{n_\pi}}{2} \cdot 2^{-|\mathsf{prek}|} + \mathsf{Adv}_4$$

Notice that we can generalize games from $\mathbb{G}_2$ to $\mathbb{G}_4$ to a single game $\mathbb{G}_{2'}$, by considering that $\mathcal{C}$ can maintain a list $\mathcal{L}_{\mathsf{DH}}$ of DH values ($\mathsf{ik}$,$\mathsf{ephk}$,$\mathsf{rchk}$,$\mathsf{prek}$) which are honestly-generated

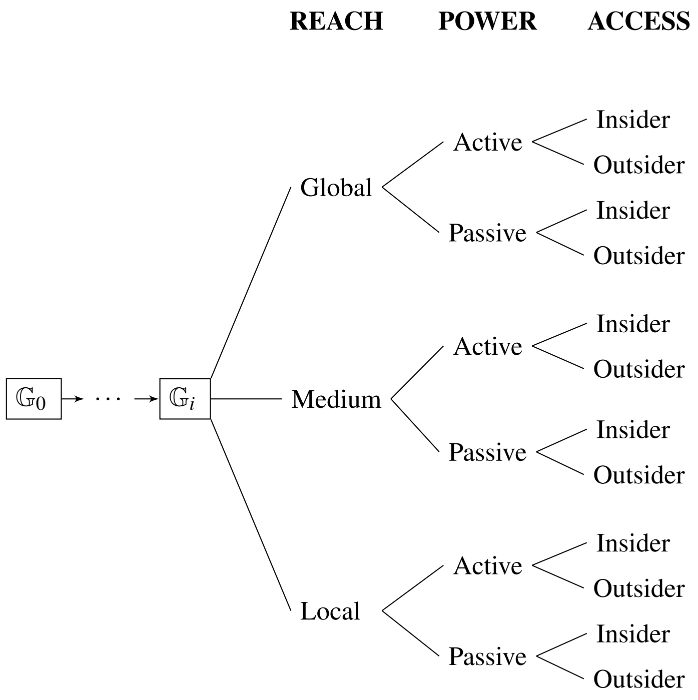


Figure 14: Overview of game hops through our taxonomy of adversary.

during the protocol. If two values appear in that list, the challenger aborts and the adversary looses the game. By considering uniqueness of identity key for each party, there are $|\mathcal{P}|$ number of identity keys, $\mathsf{n}_\pi \cdot \mathsf{n}_{\mathsf{y-max}}$ number of ratchet keys, at most $\mathsf{n}_\pi$ number of medium-term keys and $\mathsf{n}_\pi$ number of ephemeral keys. Thus we have $|\mathcal{L}_{\mathsf{DH}}| = |\mathcal{P}| + \mathsf{n}_\pi \cdot \mathsf{n}_{\mathsf{y-max}} + \mathsf{n}_\pi$. Moreover, if we consider that each DH key in $\mathcal{L}_{\mathsf{DH}}$ lies in the same group of order $q$ then a collision occurs with probability $1/q$ so we have:

$$\mathsf{Adv}_1 \leq \frac{\binom{|\mathcal{L}_{\mathsf{DH}}|}{2}}{q} + \mathsf{Adv}_{2'}$$

$\mathbb{G}_5$ : The challenger needs to guess the index $i$ of the pre-key of $Q$ used in the tested session. Since there are $\mathsf{n}_\pi$ possible values, we have:

$$\mathsf{Adv}_4 \leq \mathsf{n}_\pi \cdot \mathsf{Adv}_5$$

For clarity, we keep the notation $\mathsf{prek}_Q$ instead of $\mathsf{prek}^i_Q$ (signed pre-key of index $i$).

At this point, we will use the uniqueness and secrecy of $\mathsf{prepk}_Q^{\mathsf{rchk}_P^{0,1}}$ in order to prove indistinguishability from random of $\mathsf{rk}_1$. Note that the value $\mathsf{ms}$ (the second input of the KDF) can be learned by any reach's adversary. [11]

$\mathbb{G}_6$ : In this game the challenger accepts collision of $\mathsf{ik}_P$ and $\mathsf{prek}_P$. We need to add this condition since the next game will use a GDH challenge where the DH pair might collide with probability $1/q$, thus:

$$\mathsf{Adv}_5 \leq \frac{1}{q} + \mathsf{Adv}_6$$

Those previous games are shared between all possible adversaries of our model. We now partition our analysis given types of adversary.

**Local Passive Outsider.** In this case, the adversary can only reveal single stage keys (via the $\mathsf{oReveal.1Stage}$ oracle) in a passive way, and it has no information on the server-stored keys. Recall that (cf Fig. 4), the Signal protocol is $(\infty, 1)$-PCS secure.

$\mathbb{G}_7$ : We modify $\mathbb{G}_6$ such that the challenger aborts as soon as $\mathcal{A}$ queries the random oracle (representing the KDF) on $(\bullet \,\|(\mathsf{prepk}_Q)^{\mathsf{rchk}_P^{0,1}})$ where the first part of the input is analogous to $\mathsf{ms}$. Since our analysis is done in the random oracle model, the only way for $\mathcal{A}$ to compute the output is to give the exact input. If so, we show that when this event occur, we can construct an adversary $\mathcal{B}$ winning a GDH challenge. Recall that the GDH experiment has input $(g^a, g^b)$ to return $g^{ab}$ with a DDH oracle access with input $(g^x, g^y, g^z)$ and output 1 if $g^{xy} = g^z$.

$\mathcal{B}$ simulates $\mathbb{G}_6$ for $\mathcal{A}$ and plays against its GDH challenger. Instead of sending $g^a$ and $g^b$ to $\mathcal{A}$, it sends $\mathsf{rchk}_P^{0,1}$ and $\mathsf{prek}_Q$ respectively. Notice that $\mathcal{A}$ cannot query the $\mathsf{oReveal.1Stage}$ oracle on $\mathsf{prek}_Q$ nor $\mathsf{rchk}_P^{0,1}$ since those keys are cross-stage keys (so $\mathcal{B}$ does not have to know the private parts of those keys). However, since $\mathcal{B}$ has replaced long-term and medium-term keys of two parties (where those keys could be

[11] $\mathsf{ms}$ is a single stage key so any local, medium or global adversary can reveal it.

used in other sessions), it must ensure a valid simulation for those (non-tested) stages. In either cases, $\mathcal{B}$ randomly chooses the value $\mathsf{rk}_1$ but answers consistently with calls to the random oracle by maintaining a list. This list maps the session key with the public keys associated. Whenever $\mathcal{A}$ calls the random oracle, $\mathcal{B}$ checks if the public parts are in the list and returns the corresponding value if they are in the list, and draws a random element and adds it to the list otherwise. The special case is when $\mathcal{A}$ sends $\mathrm{CDH}(\mathsf{prepk}_Q, \mathsf{Rchpk}_P^{0,1})$ to the random oracle. In that case, the DDH oracle returns 1 when $\mathcal{B}$ queried it thus finding a solution to the GDH experiment. Finally, by noting $\varepsilon_{GDH}$ the advantage of $\mathcal{B}$ solving the GDH problem, we have:

$$\mathsf{Adv}_6 \leq \mathsf{Adv}_7 + \varepsilon_{GDH}$$

$\mathbb{G}_8$ : This game ensures the indistinguishability of the $\mathsf{rk}/\mathsf{tmp}$ outputs by the random oracle, up to and including $y^\star$. For this, we apply the modifications of $\mathbb{G}_6$ and $\mathbb{G}_7$ for a number of times equal to the maximum number of chains $\mathsf{n}_{\mathsf{y-max}}$:

$$\mathsf{Adv}_7 \leq \mathsf{n}_{\mathsf{y-max}} \cdot \left[ \binom{\mathsf{n}_\pi}{2} \cdot 2^{-|\mathsf{prek}|} + \varepsilon_{GDH} \right] + \mathsf{Adv}_8$$

Note that the same argument cannot be applied to other outputs of the random oracle (such as the chain key $\mathsf{ck}^{\cdot,y^\star}$) since those values could be revealed by the adversary (which is handled in the next game).

$\mathbb{G}_9$ : In this game, we ensure that the value $\mathsf{ck}^{0,y^\star}$ is unique. If there are two equal values in a session, or in two different (honest) sessions, then the challenger aborts and returns a random bit.

Recall that the random oracle model implies that a call to the KDF duplicates the output if the same inputs are used, or if true randomness repeats (with negligible probability), thus we have:

$$\mathsf{Adv}_8 \leq \mathsf{n}_{\mathsf{x-max}} \cdot \binom{\mathsf{n}_{\mathsf{x-max}} \cdot \mathsf{n}_{\mathsf{y-max}}}{2} \cdot 2^{-|\mathsf{ck}|} + \mathsf{Adv}_9$$

At this point, the chain key $\mathsf{ck}^{0,y^\star}$ is indistinguishable from random to $\mathcal{A}$ (which is due to the indistinguishability of $\mathsf{rk}/\mathsf{tmp}$ values from random of $\mathbb{G}_8$ ).

Depending on the adversary's reach, here local, some reveal can be queried such as the chain key $\mathsf{ck}$ or $\mathsf{mk}$. Here, the argument we used is related to the winning conditions (*i.e.*, freshness of the tested stage). Indeed, for a local passive outsider adversary, the winning conditions are parametrized by a $(\infty, 1)$ bound meaning that no $\mathsf{oReveal.1Stage}$ can be queried for a stage of index $x > 0$ and $y = y^\star - \Upsilon + 1 = y^\star$. Informally, we exclude reveal queries for stages of the same chain of the tested stage; this is a direct consequence of the symmetric ratcheting of Signal where the knowledge of one chain key implies knowledge of all the chain. Notice that our metric is also a lower bound since any strictly lower $(\chi, \Upsilon)$ value implies $\Upsilon = 0$ meaning that $\mathcal{A}$ could reveal the chain key on a stage with $y = y^\star$. This yields a trivial attack on the session keys because of the symmetric ratcheting property.

We conclude this proof by stating that:

$$\mathbb{G}_9 \leq 2^{-|\mathsf{ck}|} + 2^{-|\mathsf{mk}|}$$

Indeed, there are two possibilities for the adversary to recover $\mathsf{mk}^{x^\star,y^\star}$, either guessing directly this value (with negligible probability $2^{-|\mathsf{mk}|}$) or give as input to the random oracle the value $\mathsf{ck}^{x^\star-1,y^\star}$ (with negligible probability $2^{-|\mathsf{ck}|}$).

We have shown an upper bound of our metric in the sense that we ensure the security for at least a given number of stages. However, the security could be *faster*, *i.e.*, find a smaller metric with unchanged security. We need to show that our metric is tight meaning that we need an extra argument to show that no security can be guaranteed with smaller metric.

In this case, a local passive outsider adversary, Signal is $(\infty, 1)$-PCS secure if we exhibit an attack which compromises at least $(\infty, 1)$ stages within the adversary's type. The attack in this case is simple, $\mathcal{A}$ can reveal $\mathsf{ck}^{1,y}$, for a given $y$ on any peer, via the $\mathsf{oReveal.1Stage}$ oracle. This leads to compromising the full chain $y$ because of the symmetric ratchet deriving the keys (both $\mathsf{ck}$ and $\mathsf{mk}$). The adversary has then compromised $(\infty, 1)$ stages but no more because the next chain is initialised with cross-stage keys (*i.e.*, $\mathsf{rchk}$).

**Medium Passive Outsider.** In this case, the adversary can reveal single and cross stages keys (via the $\mathsf{oReveal.XStage}$ oracle) in a passive way, and it has no information on the server-stored keys. Recall that (cf Fig. 4), the Signal protocol is $(\infty, 2)$-PCS secure.

$\mathbb{G}_7$ : The challenger aborts if $\mathcal{A}$ gives as second input $\mathrm{CDH}(\mathsf{prepk}_Q, \mathsf{Rchpk}_P^{0,1})$ to the random oracle (the first input is a value corresponding to $\mathsf{ms}$). The keys $\mathsf{prepk}_Q$ and $\mathsf{Rchpk}_P^{0,1}$ are now in the adversary's reach possibility, via query to $\mathsf{oReveal.XStage}$ oracle. Yet, the winning conditions of this type of adversary exclude such query for a stage of chain (minimum) index $y = y^\star - 1$ for $y > 0$. So if $\mathcal{A}$ tests a stage of index $y = 1$ or $y = 2$ then the winning conditions ruled out any call to $\mathsf{oReveal.XStage}$ for such chain. As in the local case, we show that under the GDH assumption, it holds that:

$$\mathsf{Adv}_6 \leq \mathsf{Adv}_7 + \varepsilon_{GDH}$$

The reduction is the same as in the local case (where the reveal calls in the latter were excluded by the adversary's reach and by the winning conditions for the medium case). Notice that in this game, the advantage of $\mathcal{A}$ is the same as the local case, however the argument is different. For the local case, the adversary has no access to cross-stage keys while in this case, the adversary can query the $\mathsf{oReveal.XStage}$. Yet, the winning conditions for the medium case exclude such queries.

The rest of the proof is done the same way as for the local case, where in $\mathbb{G}_8$ the indistinguishability of $\mathsf{rk}/\mathsf{tmp}$

is ensured by the winning conditions (same reason as in the previous game).

We conclude the proof by showing an attack that compromised two chains since Signal is $(\infty,2)$-PCS secure for a medium passive outsider adversary. The adversary can reveal $\mathsf{rchk}^{0,2}$ and $\mathsf{rk}_1$ to get all the needed information to derive the keys of chain 2. It can also derive the $\mathtt{tmp}$ value used to initialise the next chain. So $\mathcal{A}$ has all the inputs to completely derive chain 3 (the other input to initialise the chain is $\mathsf{CDH}(\mathsf{rchk}^{0,3},\mathsf{rchk}^{0,2})$ which is computable by the adversary).

**Global Passive Outsider.** This case is actually the same as for medium case. Indeed, the argument for game hops of medium adversary implies the winning conditions for indistinguishability of keys in $\mathbb{G}_6$ and $\mathbb{G}_7$ . The attack exhibiting our metric can also be the same, while the global case could compromised the first two chains (while the medium adversary can only compromised chains starting from the second one). We can conclude that, for global passive outsider adversary, Signal is $(\infty,2)$-PCS secure.

**Local Active Outsider.** For an active adversary, we need to ensure that the keys stored on the server are generated, and signed, by the corresponding party (and not $\mathcal{A}$). Indeed, during the registration step, a party $P$ sends its identity key and pre-keys signed with the identity key. For an active adversary, some keys might be maliciously generated and sent to the server. In the case of LAO, the adversary cannot request long-term keys from its set of oracles (only ephemeral keys). Thus, we prove that the adversary needs to forge a signature.

$\mathbb{G}_7$ : Recall that in $\mathbb{G}_5$ , the challenger aborts if the chosen pre-key is different from the one used in the tested session. So the reduction to the EUF-CMA game of the signature scheme is straightforward since the challenger already knows the index of the forged signature.

For the reduction, the adversary has access to a signing oracle which updates a list of keys and signatures at each call (for avoiding trivial forgery where the signature has been already queried). We denote by $q_s$ the number of queries to this oracle. We assume here that there is an adversary $\mathcal{A}$ able to produce a valid signature on $\mathsf{prepk}$ (for a given index $i$) and we construct $\mathcal{B}$, which uses $\mathcal{A}$, to break the EUF-CMA signature scheme. $\mathcal{B}$ uses its own oracle to forward query to $\mathcal{A}$, thus:

$$\mathsf{Adv}_6 \le \mathsf{Adv}_7 + \varepsilon_{EUF-CMA}$$

From now, the following games are the same as in the local passive outsider.

**Medium/Global Active Outsider.** For those two adversaries, Signal is $(\infty,\infty)$-PCS secure meaning that no healing is possible. So we just exhibit an attack resulting in the impossibility of PCS property. The attack is simple, the adversary injects its own initial ratchet key $\mathsf{rchk}_{\star}^{0,1}$ and reveal $\mathsf{ms}$ (which is in the reach's capability of medium and global adversaries) during the initialisation phase. Thus $\mathcal{A}$ hijack the communication, where Bob is convinced to communicate with Alice, but Alice has no access to the communication (since the chain keys are different). In this case, Bob will continue the communication as long as $\mathcal{A}$ is following the protocol (it does not need to deviate from the protocol anymore).

**Passive Insider.** Each of those three types of adversaries (local, medium and global) corresponds to passive outsider adversaries. Indeed, the difference between outsider and insider is that the latter poses as the super user $\hat{S}$. In the case of Signal, this corresponds to the semi-trusted server which receives the public *bundle* keys upon registration. Because of the passive $\mathsf{access}$ type, the adversary cannot interfere with those keys so there is no difference with outsider adversary (the public keys stored by the server are also accessible by outsider adversary). For Signal, there is no difference between passive insider and passive outsider given the $\mathsf{reach}$ capability (local, medium, global). For this reason, the metric is the same for local, medium or global between outsider and insider, in the passive $\mathsf{access}$ type.

**Active Insider.** The case of active insider is the strongest type of adversary. Indeed, $\mathcal{A}$ can interfere with the protocol (*e.g.*, stop, modify messages) while compromising the server. This critical case (either local, medium or global) cannot include healing as the adversary can manipulate the keys from the start of the communication. An active insider adversary can simply remove honestly-generated keys sent to the server and replace them by its own malicious key material. In this case, the adversary plays a PiTM (Personn in The Middle) forwarding messages through Alice to Bob (and vice-versa) by its own. The communication between Alice and Bob cannot heal thus leading to a $(\infty,\infty)$-PCS secure protocol for active insider adversary.

## C.2 Games for SAID

$\mathbb{G}_0$ : This game corresponds to the original security game (Fig. 3 of Sec. 2.4). The advantage of $\mathcal{A}$ against this game is $\mathsf{Adv}_0$.

$\mathbb{G}_1$ : In this game $\mathcal{C}$ guesses $P$, $Q$, the session index of the target session, and the target stage $s^\star = (x^\star, y^\star)$ for which $\mathcal{A}$ has queried $\mathsf{oTest}$.

If $\mathcal{A}$ queries another parties, session or tested stage then $\mathcal{C}$ aborts the game and returns a random bit. Therefore we have the following:

$$\mathsf{Adv}_0 \le \mathsf{n_P}^2 \cdot \mathsf{n}_\pi \cdot \mathsf{n_{x-max}} \cdot \mathsf{n_{y-max}} \cdot \mathsf{Adv}_1$$

Moreover, we assume the uniqueness of the identity key, and identity-based related keys (identification and signature ones) for each party. The latter condition is ensured by the $\mathsf{KDC}$ maintaining a list of keys and removing possible duplicates.

**Local Passive Outsider.** We prove that SAID has the best healing, *i.e.*, $(1,0)$-PCS security meaning that only the compromised stage is accessible to the adversary. Recall that a local adversary can query the $\mathsf{oReveal.1Stage}$ oracle to reveal single-stage keys, which for SAID correspond to $\mathsf{ck}$ and $\mathsf{mk}$. First we show that the master secret $\mathsf{ms}$ is indistinguishable from random, then we show that a tested stage is fresh (with an indistinguishable session key from a random value) even after an immediate compromised stage.

$\mathbb{G}_2$ : This game aborts if the adversary calls the random oracle with input $\mathsf{ms}_{PQ}$. The adversary has only one value to guess, the random $r$ while the other values are already determined (the identity of $Q$, and the master public key of $\hat{S}$). Guessing $r$ corresponds to a large failure event so:

$$\mathsf{Adv}_1 \leq \frac{1}{q} + \mathsf{Adv}_2$$

From this game, we assume that the master secret is unique between each pair of communicating partners.

$\mathbb{G}_3$ : We show that $\mathsf{ck}^{1,y}$ is indistinguishable from random. In this game, the challenger aborts if the adversary query the random oracle with input $(\bullet \| \Delta^\star \| \mathsf{rk}^\star)$ where $\bullet$ corresponds to $\mathsf{ms}$. We show by reduction to GDH that if $\mathcal{A}$ can query the random oracle with $\Delta^\star = DH(\mathsf{Rchpk}^{0,y^\star}, \mathsf{Rchpk}^{0,y^\star+1})$ with non-negligible probability then we can construct $\mathcal{B}$ breaking the GDH problem. We apply the same technique as in Signal (cf. $\mathbb{G}_7$ ), that is $\mathcal{B}$ sends $\mathsf{Rchpk}^{0,y^\star} := g^a$ and $\mathsf{Rchpk}^{0,y^\star+1} := g^b$ to $\mathcal{A}$. If $\mathcal{A}$ does not send the query corresponding to $\Delta = \Delta^\star$ then $\mathcal{B}$ simulates completely the game for $\mathcal{A}$ while the special case is when $\mathcal{A}$ sends $\mathrm{CDH}(\mathsf{Rchpk}^{0,y^\star}, \mathsf{Rchpk}^{0,y^\star+1})$ to the random oracle. In this case, when $\mathcal{B}$ queries its DDH oracle (returning 1) it finds a solution to the GDH experiment. Finally, we have:

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \varepsilon_{GDH}$$

$\mathbb{G}_4$ : This game is the same as the previous except that the challenger aborts if $\mathcal{A}$ queries the random oracle with $\mathsf{rk}$ for up to and including $y^\star$. We use hybrid argument where the first game is $\mathbb{G}_4$ and each iteration are the next $\mathsf{rk}$ until $\mathsf{rk}^\star$. Between each game, the root keys are indistinguishable since the new root key is the output of the random oracle and $\mathcal{A}$ can only query $\mathsf{oReveal.1Stage}$. Since the adversary's probability to guess the root key is $2^{-|\mathsf{rk}|}$ for $\mathsf{n_{y-max}}$ number of chains, we have:

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + \frac{1}{2^{-|\mathsf{rk}|} - 1}$$

$\mathbb{G}_5$ : This game aborts if the adversary queries the random oracle with $(\mathsf{ck}^\star, \mathsf{ms}_{PQ}, \mathsf{tag}^\star)$. This game proceeds as the previous one with a subcase to handle. Indeed, key $\mathsf{ck}$ is in the adversary's reach (single-stage for a local adversary). Thus $\mathcal{A}$ could reveal this keys by querying oracle $\mathsf{oReveal.1Stage}$. However, as defined in 2.4, the adversary wins with non-negligible probability for a query on stage $s^\star$ which is the tested stage. Yet, the adversary has negligible probability to win if the tested stage is after the reveal query. Indeed, SAID is $(1,0)$-PCS secure meaning that $\mathcal{A}$ could reveal a key on stage $x^\star - 1$ but distinguishes the session key with negligible probability.

Suppose that $\mathcal{A}$ does not query $\mathsf{oReveal.1Stage}$ on the tested stage (which is part of our metric definition). We show by reduction that $\mathcal{A}$ can distinguish the session key if it can break the BCDH assumption meaning that it can compute $\mathsf{ms}_{PQ}$. We construct $\mathcal{B}$ simulating the game for $\mathcal{A}$. Adversary $\mathcal{B}$ receives $A = g_1^a, B = g_2^b, C = g_2^c$ as input. It sets $H(R) := A$ (with $H$ simulated as random oracle and $R$ the responder role the tested session) and $\mathsf{ID.mpk} := B$. For each other party $X \neq R$, $\mathcal{B}$ generates a random value $\alpha$ and sets $H(X) := g_1^\alpha$. When $\mathcal{A}$ starts the session between $P$ and $Q$ then $\mathcal{B}$ runs the actual protocol except that it sets $h := C$. In this case, we have $\mathsf{ms}_{PQ} = e(A,B)^c$ which is the solution of the BCDH problem instance. Observe that $\mathcal{B}$ simulates perfectly the game, except when $\mathcal{A}$ sends $\mathsf{ms}_{PQ}$ to the random oracle. Thus we have:

$$\mathsf{Adv}_4 \leq \mathsf{Adv}_5 + \varepsilon_{BCDH}$$

Finally, if $\mathcal{A}$ never sends $\mathsf{ms}_{PQ}$ to the random oracle then the session key is indistinguishable from random. In this case, $\mathcal{A}$ wins the game with probability $1/2$:

$$\mathsf{Adv}_5 = \frac{1}{2}$$

**Local Active Outsider.** This case is the same as the passive adversary except that we need to ensure that the adversary cannot replace $\mathsf{ms}$ with its own key material. Here, the adversary has two possible ways to inject its own key material. First, $\mathcal{A}$ could interfere during the registration phase between $P$ and $\hat{S}$. However, we assume that those two parties establish a secure channel thus the security relies on the AKE assumption. Second, $\mathcal{A}$ could forge its own value $h$ to compute the master secret but in this case, we rely on the EUF-CMA security of the IB-signature scheme $\mathsf{IBSig}$. Note that the active adversary case cannot interfere later on because the other keys are not single-stage (thus having the same security as the local case).

**Medium/Global Passive Outsider.** This case gathers both medium and global adversaries. Indeed, a global adversary has additional access to the keys used during registration ($\mathsf{ik}$) and identity-based key used for instance initialisation. However, in the passive this yields to no other consequence than the medium case.

SAID is $(\infty, 2)$-PCS secure meaning that the adversary can compromised two full chains of communication. We apply the same game hops as the local case, but the index of stages are different. Indeed, our security definition ensures that the call to $\mathsf{oReveal}$ cannot happen with stage $y = y^\star - 1$ or $y = y^\star$. Thus from $y = y^\star$, the adversary has the same advantage of the local case.

**Local Passive Insider.** In this case, $\mathcal{A}$ can reveal $\mathsf{ms}_{PQ}$ but not inject its own value during the protocol. We show that such adversary can compromise at most the first chain which corresponds to $(\infty, 1)$-PCS security. This is due to the fact that a new chain is initialised with ratchet keys which are out of reach's adversary.

$\mathbb{G}_2$ This game aborts if the adversary queries the random oracle with input $(\bullet \| \Delta^\star \| \circ)$ where $\bullet$ corresponds to $\mathsf{ms}$ and $\circ$ corresponds to $rk^\star$. We apply the same argument as the local passive outsider adversary in $\mathbb{G}_3$ . Thus we also use GDH reduction to show that $\mathcal{A}$. The rest of the proof correpsonds to the local passive outsider since at this point the adversary has the same advantage in both cases.

**Medium/Global Passive Insider.** In this case, SAID is $(\infty, 2)$-PCS secure. This comes directly from the fact that now the adversary has access to the secret keys used during session initialisation but cannot interfere in other way with the protocol. Our security definition implies that the adversary cannot compromise a stage of index $y = y^\star - 1$ or $y = y^\star$. We apply then the same proof as the previous case (local passive insider).

## D 5G Composed Authenticate Key-Exchanges (*5G-SCEKE*)

We present an overview outline of 5G handovers mentioned in the main document[12].

**The *Handover* Procedures: XN, our $\mathtt{XN}^+$ and N2.** The handover procedures are described mainly in [2, 3]. In this subsection, we focus only on aspects of key-establishment done within.

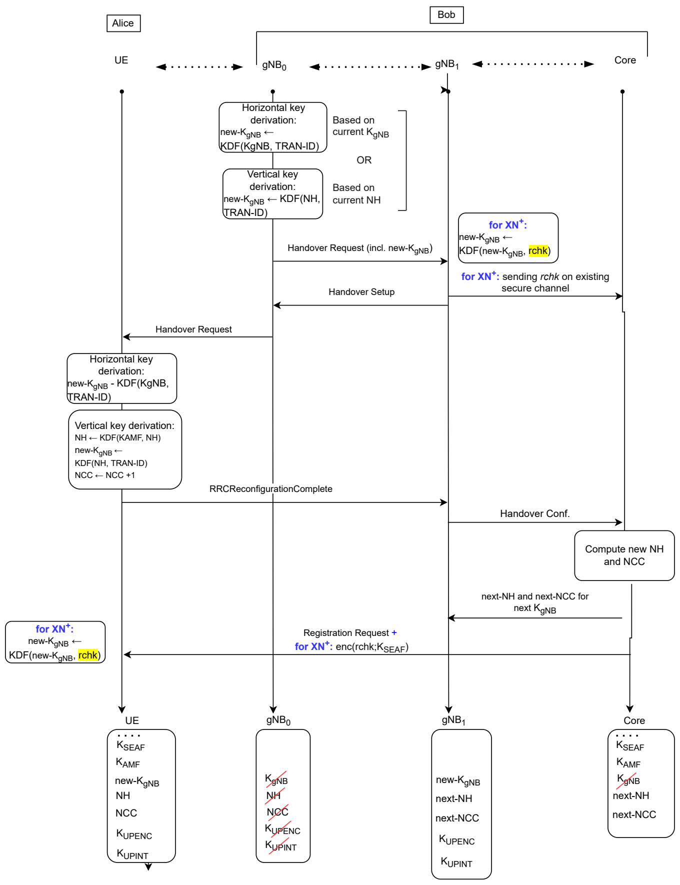
Two Types of Handovers. To communicate securely, new AS keys need to be established between the UE and the target node t-gNB. For this, as explained for the Reg procedure, a new security-key $K^1_{gNB}$ needs to arrive on the target node t-gNB. Two cases, and thus types of handovers, are possible.

1. The core generates and sends a new security-key $K^1_{gNB}$ to the target node t-gNB, similarly to the the Reg procedure; in this case, the source node is a passive proxy. This procedure is the N2 protocol.

2. The source node s-gNB, which already has a current security-key $K^0_{gNB}$ shared with the UE, is an active proxy in the procedure: it generates and sends to the target node t-gNB a new security-key $K^1_{gNB}$. This procedure is the XN protocol.

"Horizontal vs. Vertical" XN Protocol. Due to the key-derivation used to yield the application-level, so-called AS (access-stratum) keys out of the $K_{gNB}$ keys, in the XN protocol, the source node can simply use the new $K^1_{gNB}$ actually find compute the new AS keys to be used between the UE and

[12] We assume no roaming, i.e., the mobile users are served directly and entirely by the network managed of the operator they have a contract with.



Alice
Bob
UE
gNB$_0$
gNB$_1$
Core
Horizontal key derivation: new-K$_{gNB}$ ← KDF(KgNB, TRAN-ID)
Based on current K$_{gNB}$
OR
Vertical key derivation: new-K$_{gNB}$ ← KDF(NH, TRAN-ID)
Based on current NH
for XN$^+$: new-K$_{gNB}$ ← KDF(new-K$_{gNB}$, rchk)
Handover Request (incl. new-K$_{gNB}$)
for XN$^+$: sending rchk on existing secure channel
Handover Setup
Handover Request
Horizontal key derivation: new-K$_{gNB}$ - KDF(KgNB, TRAN-ID)
Vertical key derivation: NH ← KDF(KAMF, NH) new-K$_{gNB}$ ← KDF(NH, TRAN-ID) NCC ← NCC +1
RRCReconfigurationComplete
Handover Conf.
Compute new NH and NCC
next-NH and next-NCC for next K$_{gNB}$
for XN$^+$: new-K$_{gNB}$ ← KDF(new-K$_{gNB}$, rchk)
Registration Request + for XN$^+$: enc(rchk;K$_{SEAF}$)
UE
gNB$_0$
gNB$_1$
Core
K$_{SEAF}$
K$_{AMF}$
new-K$_{gNB}$
NH
NCC
K$_{UPENC}$
K$_{UPINT}$
~~K$_{gNB}$~~
~~NH~~
~~NCC~~
~~K$_{UPENC}$~~
~~K$_{UPINT}$~~
new-K$_{gNB}$
next-NH
next-NCC
K$_{UPENC}$
K$_{UPINT}$
K$_{SEAF}$
K$_{AMF}$
~~K$_{gNB}$~~
next-NH
next-NCC

Figure 15: The *5G-SCEKE* and *5G-SCEKE*$^+$ Short-term Keys' Update and Distribution at the End of XN and $\mathtt{XN}^+$($\mathtt{XN}^+$ details are in blue; red strikes denote deletion of old session keys; ... denote existence of other shared keys, and messages)

t-gNB; this is a well-known fact, and we say that XN does not have backward security. This can be worsened if the derivation of new $K_{gNB}$ is based on the previous $K_{gNB}$, which is called *horizontal key derivation (XN$^{hkd}$)*.

However, there is an alternative. At the end of each handover, the core sends to the target node a fresh key called *next-hop key (NH)*. This is derived by the core from $K_{AMF}$, which –as we explained– is refreshed (at least) at the end of each Reg execution. When a gNB is source node in an XN procedure, this gNB should not use it current $K^0_{gNB}$ to calculate a new $K^1_{gNB}$ but rather use said NH, received when/if the gNB was target in a previous execution of a handover procedure. The condition for this execution is the NH in question was not unused in this way before[13]. This type of key-derivation inside XN, whereby the new $K_{gNB}$ is not based on the previous $K_{gNB}$ but rather on a recent, core-issued NH is called *vertical key derivation (XN$^{vkd}$)*.

Clearly, in a sequence of XN executions, as soon as a vertical key derivation takes place, the chain of serial loss of

[13] The NH could have been used by the current s-gNB, if this gNB received it at the end of N2, or if the served UE had temporarily been in IDLE state.

backward security is broken. Yet, if someone knows the NH on then this lack of security of the future $K_{gNB}$ continues.

In this work, we consider XN with either type of key derivation for $k_{gNB}$ (corresponding to $\mathsf{ck}$ in SCEKE).

The XN$^+$ Protocol. To improve the backwards security in XN, in the main body, we proposed *XN*$^+$. Simply put, we added a randomness "rchk" that the t-gNB adds to each $k_{gNB}$ calculation. And this randomness is securely sent to the core which sends it to the UE, in turn, encrypted with $k_{SEAF}$; all these are piggybacked on existing messages. See the yellow markers in Figure 15.

The N2 Protocol. The N2 protocol requires less trust in the gNBs, since the source node is not actively calculating the security-key for the target node. In N2, the derivation of $K^*_{gNB}$ is in fact a vertical key derivation based on a new NH locally computed by the core and sent to t-gNB.

What is more, the core can be configured to re-fresh the keys further up the key hierarchy, all the way to recomputing $K_{AMF}$, which we aforementioned as the bottom of the AKA-refreshed key chain; this is referred to as *" N2 with $K_{AMF}$ rekeying"*. It is this latter type of N2 that we use in this work inside *5G-SCEKE* and *5G-SCEKE*$^+$.

The choice between the one type of handover or another, i.e., XN vs N2, is down to the interfaces between s-gNB and t-gNB: if there is an XN interface between them, then XN is executed; otherwise, N2 will be executed, intermediated by the *core*.

## E Security of *5G-SCEKE* and *5G-SCEKE*$^+$

### E.1 Security of *5G-SCEKE* as SCEKE

We now explain the results given in Figure 4 for *5G-SCEKE*. In the analysis below, we consider all relevant attackers in our framework

GLOBAL AND MEDIUM ADVERSARIES *5G-SCEKE*'s PCS-security of a medium and global adversary (outsider or insider and active or passive) is $(\infty,\infty)$. For this attacker, the reveal of one root key $\mathsf{K}_{AMF}$ leads to having access to the rest of the communication, after the reveal. In the key hierarchy, each message key from stage (x,y) can be computed from $\mathsf{rk}_y$, or a fortiori from the long-term key $\mathsf{K}$.

OTHER INSIDER ADVERSARIES The protocol in this case is $(\infty,\infty)$-PCS-secure. For this attacker, no healing is possible since the attacker has access to keys "above" $\mathsf{K}_{AMF}$ in the 5G hierarchy, i.e., above $\mathsf{rk}$.

The above statements on *5G-SCEKE* are rather attacks, so no formal proof is needed.

LOCAL OUTSIDER In this case, *the* 5G-SCEKE *protocol is* $(\infty,1)$*-PCS-secure*. To this end, see that the chain and message keys are derived symmetrically, and that $\mathsf{rk}$ is not revealed from a call to oReveal.1Stage. So, $\mathcal{A}$ has no access to stage (1,y+1). Viewed differently, $(\infty,1)$-PCS-security in this case can be explained via the fact the horizontal evolution is the same as in Signal.

The security statement accompanying the healing related tqo the latter attack above is given below.

**Theorem 4** *: Consider the* 5G-SCEKE *protocol modelled as a SCEKE scheme in our model. The following results hold in the random oracle model (by replacing the KDFs with random oracles):*

- 5G-SCEKE *is* $(\infty,1)$*-PCS secure against local outsiders (active or passive).*
- 5G-SCEKE *is* $(\infty,\infty)$*-PCS secure against insiders, medium and global outsiders.*

We elude the proof here as this is very similar to the proofs for *5G-SCEKE*$^+$, which follow.

Indeed, we continue with *5G-SCEKE*$^+$, for which we first show security informal, and also accompany this by formal proofs.

### E.2 Security Proofs for *5G-SCEKE*$^+$ as SCEKE

In the main body of the document, we gave Theorem 3 for the security of *5G-SCEKE*$^+$. We note that, in order to prove those statements, we must provide two elements for each type of attacker: a proof of security for the stages for which the security does heal, and an attack that breaks the security of the remaining stages. We first describe the attacks, informally.

PASSIVE OUTSIDER This case captures the gain from our modification. Indeed, $\mathcal{A}$ can have access to a specific stage through any combination of keys, but it cannot attain anything from a next stage since a fresh value $\mathsf{rchk}^{x,y}$ is used for the next stage. This entails to $(1,0)$-PCS-security, in this case.

LOCAL ACTIVE OUTSIDER In this case, $\mathcal{A}$ can recover the material of a stage, but cannot send its own value $\mathsf{rchk}^{x,y}$, since $\mathsf{K}_{AMF}$ (*i.e.*,$\mathsf{rk}$) is needed first. Thus it equates to the previous case, leading to $(1,0)$-PCS-security.

OTHER ACTIVE OUTSIDER ADVERSARIES Now $\mathcal{A}$ has access to the key $\mathsf{K}_{AMF}$ leading to the hijack of the communication. $\mathcal{A}$ can generate its own value $\mathsf{rchk}^{x,y}$ and sends it to the intended partner. Thus $\mathcal{A}$ take full control of the communication over the compromise party. This corresponds to $(\infty,\infty)$-PCS-security.

INSIDER ADVERSARIES Here no healing is possible, the protocol is $(\infty,\infty)$-PCS-secure. The adversary has access to the top key in the hierarchy considered in *5G-SCEKE*$^+$, meaning that it can compute all the keys and uses its own key material.

**Proofs.**

Theorem 3: *"Consider the* 5G-SCEKE$^+$ *protocol modelled as a SCEKE scheme, in our model. The following results hold in the random oracle model (by replacing the KDFs with random oracles)*

- 5G-SCEKE$^+$ *is* $(1,0)$*-PCS secure against local active outsiders.*
- 5G-SCEKE$^+$ *is* $(1,0)$*-PCS secure against passive outsiders (local or global);"*

- *For all other adversary types,* 5G-SCEKE$^+$ *is* $(\infty,\infty)$-*PCS secure (i.e., the security of the channel never heals upon compromise).*

**Proof.**

$\mathbb{G}_0$ : This game corresponds to the original security game (Fig. 3 of Sec. 2.4). The advantage of $\mathcal{A}$ against this game is $\mathsf{Adv}_0$.

$\mathbb{G}_1$ : In this game, the challenger $\mathcal{C}$ guesses $P$, $Q$, the session index of the target session, and the target stage $s^\star = (x^\star, y^\star)$ for which $\mathcal{A}$ has queried $\mathsf{oTest}$.

If $\mathcal{A}$ queries another parties, session or tested stage then $\mathcal{C}$ aborts the game and returns a random bit. Therefore, we have the following:

$$\mathsf{Adv}_0 \leq {\mathsf{n_P}}^2 \cdot \mathsf{n}_\pi \cdot \mathsf{n_{x-max}} \cdot \mathsf{n_{y-max}} \cdot \mathsf{Adv}_1.$$

The next game deals with the fact that the locally generated secrets $\mathsf{rchk}^{x,y}$ in $\mathtt{XN}^+$ do not collide.

$\mathbb{G}_2$ : This game is the same as $\mathbb{G}_1$ except that the challenger aborts if two keys $\mathsf{rchk}$ (i.e., two nonces $\mathsf{rchk}^{x,y}$ used in $\mathtt{XN}^+$), used inside the derivation of $\mathsf{ck}$s (i.e., $k_{gNB}$s), have a collision. We have that the advantage in $\mathbb{G}_2$ is:

$$\mathsf{Adv}_1 \leq \binom{\mathsf{n}_\pi}{2} \cdot 2^{-|\mathsf{rchk}|} + \mathsf{Adv}_2.$$

$\mathbb{G}_3$ : This game is the same as $\mathbb{G}_4$ except the possibility of distinguishing a RO in place where the KDF computing the keys $\mathsf{rk}$ s (i.e., $K_{AMF}$ is used) is used.

$$\mathsf{Adv}_2 \leq Adv^{RO(KDF_{K_{AMF}})} + \mathsf{Adv}_3.$$

At this point, all the keys $\mathsf{rk}$ (i.e., the $k_{amf}$s) are indistinguishable from random based on the hypothesis of KDFs being replaced by ROs.

In fact, we apply the modifications of $\mathbb{G}_1$ and $\mathbb{G}_3$ a number of times equal to the maximum number of chains $\mathsf{n_{y-max}}$, and we call the result game $\mathbb{G}_4$ . We have that the advantage in $\mathbb{G}_4$ is:

$$\mathsf{Adv}_3 \leq \mathsf{n_{y-max}} \cdot \left[ \binom{\mathsf{n}_\pi}{2} \cdot 2^{-|\mathsf{rchk}|} + Adv^{RO(KDF_{K_{AMF}})} + \mathsf{Adv}_2 \right] + \mathsf{Adv}_4$$

$\mathbb{G}_5$ : This game is the same as $\mathbb{G}_4$ except the challenger aborts if the keys $ck^{(x,y/y')}$ are the same, for two distinct chains of stages $y$ and $y/y'$ of the same session or of two different honest sessions.

Note that, since at this stage the $\mathsf{rchk}^{x,y}$s are non-colliding, all the keys $\mathsf{ck}$ (i.e., the $k_{gNB}$s) as the above are unique until the randomness repeats itself (as the KDF producing $\mathsf{ck}$ is considered a random oracle). So, we have:

Also, the chain keys $\mathsf{ck}^{0,y^\star}$ are indistinguishable from random to $\mathcal{A}$, which resides on the fact that the KDF yielding $\mathsf{ck}^i$s considered an RO (as well as from the uniqueness of $\mathsf{rchk}$ values in $\mathbb{G}_2$ , and from the randomness of $\mathsf{rk}$ values in $\mathbb{G}_3$ ).

So, we have:

$$\mathsf{Adv}_4 \leq Adv^{RO(KDF_{K_{gNB}})} + \mathsf{n_{x-max}} \cdot \binom{\mathsf{n_{x-max}} \cdot \mathsf{n_{y-max}}}{2} \cdot 2^{-|\mathsf{ck}|} + \mathsf{Adv}_5$$

**Adversary-types' Dependencies & Winning Conditions.**
**1.** In the case where our adversary is **local**, reveal queries could get the adversary to know chain-keys such $\mathsf{ck}$ or $\mathsf{mk}$.

Next, the argument is based, in part, on the fact that in our framework any proven bound strictly lower $(\chi, \Upsilon)$, also implies the bound $(\chi, 0)$, i.e., $\mathcal{A}$ could reveal the chain-key on a stage with $y = y^\star$. Let us assume therefore that the reveal is made on a stage with $y = y^\star$.

In this case, the **local** attacker gets $\mathsf{ck}$ (i.e., $k_{gNB}$) on a stage with $y = y^\star$, but to compute a new $\mathsf{ck}$ and $\mathsf{mk}$ needs the following:

- the next $\mathsf{rchk}$ (i.e., the $\mathsf{rchk}^{x,y}$) value – the gNB-driven randomness that we added to $\mathtt{XN}^+$, which the attacker cannot introduce.

**2.** In the case where our adversary is **global**, reveal queries could get the adversary to know keys such as $\mathsf{rk}$.

In this case, the **global passive** attacker even get $\mathsf{rk}$ (i.e., $k_{AMF}$), but we fall back to the case above: that is, to compute a next $\mathsf{mk}$ (i.e., $k_{AS}$), the **global passive** attacker still needs the following:

- The next $\mathsf{rchk}$ (i.e., the $\mathsf{rchk}^{x,y}$) value – the gNB-driven randomness that we added to $\mathtt{XN}^+$, which the attacker cannot introduce.

We conclude this proof: for the case **global passive** and **local active** attacker, *5G-SCEKE*$^+$ is (1,0)-PCS secure, with the final advantage of such adversaries being $\mathbb{G}_5 \leq 2^{-|\mathsf{ck}|} + 2^{-|\mathsf{mk}|}$, i.e., these adversaries can just guess the $\mathsf{ck}$ and/or $\mathsf{mk}$s, after the crucial "reveal"s have been called, and the probability of such guessing is negligible.