Université Clermont Auvergne
Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS)
CNRS, UMR 6158, LIMOS, 63173 Aubière, France

Design and Analysis of Provably Secure Protocols: Applications to Messaging and Attestation

Thèse

pour obtenir le

Doctorat de L'Université Clermont Auvergne

Spécialité : Informatique

École Doctorale des Sciences Pour l'Ingénieur

présentée et soutenue publiquement par

LÉO ROBERT

le 22 septembre 2022

devant le jury composé de :

M. Benjamin NGUYEN	Professeur des universités LIFO, INSA Centre Val de Loire	Rapporteur
Mme Melek ÖNEN	Maître de conférences Eurecom, Sophia Antipolis	Rapporteure
M. Karthikeyan BHARGAVAN	Directeur de Recherche INRIA, Paris	Examinateur
M. Jean-Guillaume DUMAS	Professeur des universités LJK, Université Grenobles Alpes	Examinateur
M. David POINTCHEVAL	Professeur des universités ENS, INRIA, PSL, Paris	Président du jury
M. Olivier SANDERS	Ingénieur de recherche Orange Labs, Cesson-Sévigné	Examinateur
M. Pascal LAFOURCADE	Maître de conférences LIMOS, Université Clermont Auvergne	Co-Directeur de thèse
Mme. Cristina ONETE	Maître de conférences XLIM, Université de Limoges	Co-Directrice de thèse

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 endpoints 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 features in prominent messaging protocols such as Signal.

In this thesis, we first present two variants of Signal which improve the PCS property. Moreover, by viewing PCS as a spectrum, rather than a binary property which schemes might or might not have, in the second part of the thesis 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 and our variant SAMURAI, but also an identity-based messaging protocol named SAID, and even a composition of 5G handover protocols. We also apply the results obtained for this last example in order to provide a quick fix, which massively improves its post-compromise security.

The last part of this thesis is dedicated to the question of deep attestation in virtualized infrastructures. Deep attestation is a particular case of remote attestation, *i.e.*,verifying the integrity of a platform in the presence of a remote server. We focus on the remote attestation of hypervisors and their hosted virtual machines (VM), for which two solutions are currently supported by ETSI (European Telecommunications Standards Institute). The first is single-channel attestation, requiring for each VM an attestation of that VM and the underlying hypervisor through the physical TPM. The second, multi-channel attestation, allows to attest VMs via virtual TPMs and separately from the hypervisor – this is faster and requires less overall attestations, but the server cannot verify the *link* between VM and hypervisor attestations, which is naturally available for single-channel attestation.

We design a new approach which provides linked remote attestation which achieves the best of both worlds: we benefit from the efficiency of multi-channel attestation while simultaneously allowing attestations to be linked. Moreover, we formalize a security model for deep attestation and *prove* the security of our approach. Our contribution is agnostic of the precise underlying secure component (which could be instantiated as a TPM or something equivalent) and can be of independent interest.

RÉSUMÉ

La Sécurité Après-Compromission (PCS pour *Post-Compromise Security*) est une propriété de sécurité concernant les schémas d'établissement de canal sécurisé. Elle vise à limiter les failles de sécurité que pourrait introduire un attaquant en compromettant un utilisateur. Le phénomène de guérison, qui est le résultat de la PCS, permet d'*éjecter* l'attaquant ce qui rend le canal à nouveau sécurisé. Cette propriété intéressante, surtout depuis les révélations d'Edward Snowden concernant la surveillance de masse, se retrouve dans la plupart des protocoles de messageries populaires, notamment Signal.

Dans cette thèse, nous présentons dans un premier temps deux variantes de Signal; ces deux protocoles (MARSHAL et SAMURAI) sont construits pour améliorer la propriété de PCS. En faisant l'observation que la PCS n'est pas une propriété binaire mais plutôt un spectre de possibilités, nous proposons dans un second chapitre un modèle pour quantifier et comparer la PCS en fonction du type d'adversaire considéré. Nous détaillerons d'ailleurs tous les adversaires possibles pour notre contexte. La généralité et flexibilité de notre approche nous permet de modéliser une vaste diversité de protocoles, en particulier Signal mais aussi une nos variantes décrite dans le premier chapitre, ainsi qu'une autre variante (SAID) de Signal mais basée sur l'identité. Un dernier cas est analysé, montrant l'expressivité de notre modèle, celui d'une série de procédure pour le réseau 5G nommée protocoles relais (*handover*). L'étude de ce dernier cas nous amène à proposer une amélioration concernant la PCS pour un protocole de relais.

La dernière partie de cette thèse se concentre sur un problème lié à la sécurité, l'attestation dans le contexte de virtualisation. L'attestation en profondeur, un type d'attestation, permet de vérifier l'intégrité d'une plateforme à l'aide d'un serveur de vérification à distance. Nous nous concentrons sur l'attestation d'hyperviseurs et de machines virtuelles. Il existe deux solutions standardisées, la première permet d'attester l'hyperviseur et la machine virtuelle en même temps alors que la deuxième permet d'attester indépendamment ces deux composants. Nous proposons une solution qui regroupe les avantages de ces deux alternatives (sécurisée et efficace) dans un modèle inédit. Le but est de formaliser un modèle de sécurité visant à prouver la sécurité de notre approche.

REMERCIEMENTS

Mes premiers remerciements vont tout naturellement à mes deux encadrants de thèse, Pascal LAFOURCADE et Cristina ONETE.

Tout d'abord, merci Pascal. Tu m'as fait décrouvrir le monde de la recherche, ta passion et ta rigueur m'ont stimulé tout au long de mon parcours. Tu m'as permis d'être autonome lors de ces trois années, tout en te rendant disponible dès que j'en ai eu besoin. C'est en grande partie grâce à toi que j'ai apprécié chaque instant de cette thèse, avec une mention spéciale pour tous ces moments, formidables mais fugaces, de recherche. Je pense, en particulier, à ces deux jours en début de thèse passés à Limoges dans un bureau avec Cristina et Olivier (Blazy) où j'ai assisté à des échanges d'idées qui fusaient sans cesse. J'espère avoir de nouveau l'occasion de participer à ces moments avec toi.

Merci Cristina. Ta rigueur, ton expertise en cryptographie et ta fascinante qualité d'écriture ont été, et sont encore pour moi, une source d'inspiration. Tu m'as toujours apporté une aide précise et pertinente sur les difficultés techniques auxquelles j'ai souvent été confronté. Tu as su me transmettre tes connaissances en cryptographie; j'ai beaucoup appris à tes côtés et j'espère pouvoir continuer à le faire.

Je tiens aussi à remercier Benjamin NGUYEN et Melek ÖNEN pour le sérieux, la rigueur et l'interêt dont ils ont fait preuve pour rapporter cette thèse. Je remercie également Karthikeyan BHARGAVAN, Jean-Guillaume DUMAS, David POINTCHEVAL et Olivier SANDERS pour avoir bien voulu faire partie des membres du jury.

J'ai un remerciement tout particulier à adresser à Jean-Guillaume, qui fut mon encadrant de stage de master. C'est grâce à lui que j'ai pu réaliser mes premières expériences de recherche. C'est aussi grâce à lui, qui m'a parlé du sujet de cette thèse et m'a conseillé de candidater, que je me suis lancé dans cette aventure.

Contents

INTRODUCTION

In this work, we focus on a specific field of science, called cryptography. While early forms of cryptography (also referred to as "classical cryptography") featured a race between a cryptographic design and ever-evolving attacks against it, the modern approach is to use scientific methods which enforce formal security definitions and a rigorous approach. A correlated event marking the evolution from classical to modern cryptography is the rise of computer science initiated by Alan Turing. The democratisation of computers (as an object) has fuelled a virtualisation of interactions and processes in our daily lives, such as communications, payment, elections, etc.

It is crucial to understand that no security for such technologies can be absolute and universal. Even powerful methodologies, such as provable security or automated verification, can only provide a partial understanding of the properties achieved by a cryptographic primitive or protocol in a given setting. Even then, the guarantees provided only hold if the remaining components involved in the deployment of the primitive or protocol (*e.g.*, concrete implementations, OS characteristics, or even the user's behaviour) follow a certain pattern, or model. However, by providing at least this partial rigorous formalism, we can ensure that the aspect of the primitive that we do examine – in our case, its cryptographic design – is sound, and therefore no attacker can exploit it to its nefarious purposes. In particular, in this thesis we focus on the provable security of cryptographic protocol design.

One aspect described in this work is the establishment of a secure channel between two parties. How can they securely communicate and how much trust should they put in the channel? While those questions are critical, we must also stop a moment and ask ourselves why it is important to secure data. While answering this question is clearly out of reach in a purely scientific approach (as it also includes social, political and economical aspects), we can observe a paradigm shift caused by Snowden's revelations. There was a before, where data security was not a concern for the majority of citizens, and an after where mass surveillance by governmental agencies became a known fact and everybody's concern. This is how security for everyday services became a perceived necessity, especially for messaging protocols.

In this work, we propose a partial answer to how to improve security for such protocols.

Trevor Perrin and Moxie Marlinspike designed Signal, an asynchronous messaging protocol (in 2013). They received the Levchin prize in 2017 for their work itself and also for the impact on society it brought. Signal enlarged the possibility of using secure applications to communicate at a large scale. Numerous applications used Signal to provide end-to-end encryption, for instance WhatsApp, Facebook Messenger, Skype (through the private conversations), etc ..., making Signal the go-to solution for a high level of security in asynchronous messaging. This situation is reinforced by the Message Layer Security (MLS) development by both industrial and researchers. The goal is to retrieve all the qualities of

Signal to use it in group messaging (as Signal has only pairwise communications). The main security properties for messaging protocols are provided using Signal, for example the asynchronous property (Alice and Bob need not be online at the same time to communicate), confidentiality (only Alice and Bob know the content of each message), authentication (Alice is sure she is talking to Bob and vice-versa), or Perfect-Forward Secrecy (the communication before a compromise is secure). Yet, Signal is known to pioneer *Post-Compromise Security*, an unprecedented property allowing some guarantee against an active attacker.

The PCS property has become a must-have in asynchronous messaging, since messaging sessions are much longer than in typical secure-channel establishment – thus, on the one hand, the security of each channel is much likelier to be compromised during its lifetime, and on the other hand, the consequences could be much more serious Post-compromise security allows a compromised channel to recover its original security, even after the latter is downgraded by potential attacks. If an attacker has access to a communication by revealing some secrets (but not all) then the protocol eventually recovers from it and *heals*. The channel's healing returns the attacker to the position of an outsider, thus removing the important threat of passive mass surveillance. In other words, in order to continue learning information from the previously-compromised channel, the adversary has no choice but to renew its active threat.

Two questions naturally arises from those observations:

- *Can we quantify the interval that passes between the moment of compromise and the subsequent healing?*
- *Can we improve the PCS notion of Signal?*

Two chapters are dedicating to answering (in the affirmative) these questions. First, we propose two protocols, MARSHAL and SAMURAI, then our generic approach to analyse PCS notion.

MARSHAL and SAMURAI. We propose two variants of Signal. The latter has an interesting property called Post-Compromise Security (PCS) which enables the protocol to *fight* against an attacker. This phenomenon, called healing, occurs when an attacker manages to reveal some (but not all) secrets about a communication between two honest parties. The PCS ensures that, at some point after the compromise, the protocol meets its security again. The main idea behind PCS is key evolution.

There are two main issues for the context of messaging protocols. First, majority of protocols (like Signal) are proposing asynchronous feature meaning that Alice and Bob need not be online at the same time in order to communicate. This means that the key evolution must be also asynchronous in the sense that no strong interactivity can be considered (for instance sigma protocols are interactive; we cannot include such kind of process in messaging protocols without loosing the asynchronous feature). For Signal, the key evolution is computed with Diffie-Hellman values allowing some sort of non-interactivity.

Yet, the key evolution must be computed by correct parties. This corresponds to the second issue of PCS in messaging protocols context. By allowing the session keys to evolve within the communication, some malleability is added throughout the session. The keys are

evolving but by which party (or parties for group messaging)? It is critical to ensure that a computation is done by authorized parties and not an attacker. Letting the cryptographic guarantees evolve leads the potential attackers to have a wider range to inject flaws into the communication (compared to a constant value). One solution to this problem is the persistent authentication developed by Blazy *et al.* for their SAID protocol, an identity based Signal-like protocol.

Our two protocols, MARSHAL and SAMURAI are mainly designed around those two issues. The key evolution is done for each session keys (message keys in our concern) instead of speaker alternation (*e.g.*, Alice sends the messages then this Bob sending messages). Thus we compute more frequently the evolution of keys than in Signal. This approach is better in terms of security but the performances are altered although the running times remain practical. We also add the persistent authentication in our protocols to avoid communication hijack (meaning that the attacker completely impersonates one party to the others). Notice that the security is strengthened thanks to the persistent authentication but, in the meantime, this inhibits any deniability feature. So we increase security albeit privacy is the cost.

Our approach in both protocols is to stay close to Signal design. Unlike other works with modular design, our protocols give straightforward comparison to Signal and highlight the crucial components for enhancing PCS (as given in the previous paragraph, *i.e.*, the frequency of asymmetric ratchet and persistent authentication).

Although MARSHAL and SAMURAI have the same level of security in terms of PCS (also for others but this is not our main concern here), they differ by their performances. Surprisingly, their differences are not the cause of PCS issue but from another property called out-of-order. Our approach (sticking close to Signal) makes the design of variants tricky because several security properties need to be guaranteed while PCS is improved. Thus, changing some part of the protocol requires to balance some other parts in order to keep the same level of security or usability. In our case, the out-of-order messages directly impacts the key derivation (especially the message keys) because some messages could be lost during the communication. Keeping this feature while ensuring an asymmetric ratchet implies to keep some information from previous stages. In MARSHAL, for a given chain (*i.e.*, without changing the speaker), some of the auxiliary data [1] of all the previous stages are appended to the next stage. This ensures the receiver to always being able to recover that session key even if some stages are lost in between. The major drawback is the size of the auxiliary data which grows linearly in the size of a chain. Yet, SAMURAI overcomes this problem by deriving independently the chain key (by symmetric ratchet) and the message key (by asymmetric ratchet). This implies that the auxiliary data is minimal for each stage because there is only information about that very stage (and not from the previous ones) while ensuring out-of-order messages.

The difference between MARSHAL and SAMURAI does lie in their memory management of the associated data for messages inside a chain; although their respective security are identical (especially for PCS).

The work on MARSHAL can be found in:

[1] those are associated data of the AEAD scheme which are needed for the receiver in order to derive the key session, and also for other security properties like authentication

Olivier Blazy, Pierre-Alain Fouque, Thibaut Jacques, Pascal Lafourcade, Cristina Onete, and Léo Robert. MARSHAL: messaging with asynchronous ratchets and signatures for faster healing. In Jiman Hong, Miroslav Bures, Juw Won Park, and Tomás Cerný, editors, *SAC '22: The 37th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, April 25 - 29, 2022*, pages 1666–1673. ACM, 2022

A generic framework to quantify PCS. The previous chapter is dedicated to improve a protocol for a specific security property, but how can we formalize and quantify such improvement? This question is trivial for binary (in the sense yes/no output) properties but tricky for properties with unclear process. For instance, considering the Perfect Forward Secrecy (PFS) is pointless to propose an improvement. Indeed, the PFS ensures that previous communications from a compromise is secure, this is a plain result meaning that some protocols have this property and some do not, there is no in-between. Yet, the PCS is interesting to analyse because of its wide spectrum of application. A protocol with PCS feature will (conditionally) recover its security after a compromise but this will occur at some point in the future. The essence of those properties implies that there is no clear timeline within their activation. So some protocols have or have not the PCS property but we need to analyze deeper for protocols having it.

We propose in this chapter such analysis for PCS property. The goal is to be able to compare protocols having the PCS. The results obtained are manifold.

First, we can compare protocols by giving a framework for secure-channel establishment with key evolution protocols. Our generic approach allows the comparison of protocols which are, at first glance, incomparable. We embed our model with a metric that gives a quantification to the PCS. A major interest is the possibility to deduce which protocol is better.

We do not cover all the protocols featuring PCS. Yet, we adopt a specific strategy to ensure the liability and accuracy of our model. We choose to analyse first Signal which is well analysed in the literature and used in numerous messaging protocols. Our model is based on this protocol as the notion of PCS is crucial for it. Then we analyse SAMURAI and SAID, which both are variants of Signal. While SAMURAI is described in the previous chapter and close to Signal, the other variant SAID lies in a different framework.

Indeed, SAID is an identity-based protocol. The concept of identity-based cryptography was introduced by Shamir in [Sha84] with the purpose of diminishing our reliance on public-key infrastructure. In this paradigm, there is an authority (*e.g.*, a Key Distribution Center) that generates keys for users. The principle of identity-based cryptography lies in associating users with their short, easy-to-remember identities (*e.g.*, an email address or a phone number). The first instantiations of identity-based encryption were presented in [BF03], then signature schemes [BNN09] and Authenticated Key Exchange (AKE) protocol [JHJ05].

Finally, we show the expressiveness of our model with protocols that are completely different (*i.e.*, the 5G handover procedures). The analysis conducted brings a variant we designed, which improves existing standard.

We also give a taxonomy of adversaries. The results obtained by our metric are classified by types of adversaries. The latter are a mix of known adversaries in the literature and new

ones from our analysis. We consider adversaries composed of three characteristics: the *reach* (a fine-grained classification depending on the possible keys to be revealed), the power (passive or active), the access (insider or outsider). By combining those components, there are 12 possible adversaries giving 12 results for PCS for each protocol.

The results from our metric give also a deeper understanding on the mechanisms composing the PCS. From the 5G handover analysis, we observe that key evolution in a deterministic manner (*i.e.*, symmetric ratchet) is disastrous for healing. It simply comes from the fact that key derivation can be computed efficiently in one way. Thus it is crucial to add some unexpected values (*i.e.*, random elements used for computing Diffie-Hellman values) to break the routine. An other aspect is the frequency of such non-deterministic computations. As an example, consider Signal with SAMURAI where the latter has asymmetric ratchet for each stage compared to only when the speaker changes for the former. Finally, we also observe the effect of persistent authentication (*i.e.*, signing auxiliary data along each message) for active adversaries.

This work will appear in the Usenix conference:

Olivier Blazy, Ioana Boureanu, Pascal Lafourcade, Cristina Onete, and Léo Robert. How fast do you heal? A taxonomy for post-compromise security in secure-channel establishment. *Usenix Security Symposium*, page to appear, 2023. `https://eprint.iacr.org/2022/1090`

Authorized Linked Attestation. This chapter has a different context from the previous ones. We focus on attestation, which is the process where a verifier can check if a property of an entity is correct. In virtualized infrastructures, both software and hardware parts are entangled and a variant of attestation called deep attestation ensures that both sides are attested. The components we consider are threefold, a root of trust (hardware part), virtual machines and hypervisors managing them. The goal of deep attestation is thus to attest a virtual machine and the hypervisor with the root of trust.

We propose a solution which balances two existing solutions. Indeed, the single-channel attestation is secure but poorly scalable while the multi-channel allows attestation at large scale but with little guaranty toward security. Our solution is both secure and scale efficiently. This assertion is reinforced by two arguments, first we propose a formal treatment of our new protocol to prove its security. Second, we show the performances of our construction with a proof-of-concept. So our approach shows confidence both in theoretical and practical aspects.

The model we designed to construct our solution in deep attestation context is the first in the literature, and solutions already existed but not formally proved. We choose composition-based approach where sequence of increasingly stronger primitives are layered to obtain our authorized linked attestation. The goal is to attest each component individually by an authorized entity, and being able to link those attestations.

This solution is straightforward but the treatment we made is hard since attestation is a generic term regrouping numerous possible applications which have different goals. Despite being practical, we believe that our construction within its formal treatment can be of independent interest to this line of research.

This work appears in:
Ghada Arfaoui, Pierre-Alain Fouque, Thibaut Jacques, Pascal Lafourcade, Adina Nedelcu, Cristina Onete, and Léo Robert. A cryptographic view of deep-attestation, or how to do provably-secure layer-linking. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*, volume 13269 of *Lecture Notes in Computer Science*, pages 399–418. Springer, 2022

Outline of this manuscript. We first introduce the basic notions needed throughout this work in Chapter 1. Then, in Chapter 2, we present two protocols (MARSHAL and SAMURAI) which improved the level of PCS achieved by Signal. The next part, in Chapter 3, describes a more generic approach to PCS and broaden the use case of Signal. Chapter 4 focuses on the topic of deep attestation, for which security is formalized, and then proved for a particular scheme of our design. We also provide a short summary in French of the findings of this thesis in Appendix A.

Other publications. We list some publications from other works that are not developed in this thesis.

The subjects of those published papers are card-based ZKP protocols, physical secret sharing, generic construction of signature scheme, improvements of public-key schemes, and finally an unplugged activity to show that perfect anti-virus does not exist. We give an abstract of each publications. Note that the list is not chronological but sorted by topic.

- Daiki Miyahara, Léo Robert, Pascal Lafourcade, So Takeshige, Takaaki Mizuki, Kazumasa Shinagawa, Atsuki Nagao, and Hideaki Sone. Card-Based ZKP Protocols for Takuzu and Juosan. In Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara, editors, *10th International Conference on Fun with Algorithms (FUN 2021)*, volume 157 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:21, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik

  Abstract: Takuzu and Juosan are logical Nikoli games in the spirit of Sudoku. In Takuzu, a grid must be filled with 0's and 1's under specific constraints. In Juosan, the grid must be filled with vertical and horizontal dashes with specific constraints. We give physical algorithms using cards to realize zero-knowledge proofs for those games. The goal is to allow a player to show that he/she has the solution without revealing it. Previous work on Takuzu showed a protocol with multiple instances needed. We propose two improvements: only one instance needed and a soundness proof. We also propose a similar proof for Juosan game.

- Léo Robert, Daiki Miyahara, Pascal Lafourcade, Luc Libralesso, and Takaaki Mizuki. Physical zero-knowledge proof and np-completeness proof of suguru puzzle. *Inf. Comput.*, 285(Part):104858, 2022

  Abstract: Suguru is a paper and pencil puzzle invented by Naoki Inaba. The goal of the game is to fill a grid with numbers between 1 and 5 while respecting three simple constraints. We first prove the NP-completeness of Suguru puzzle. For this we design

gadgets to encode the PLANAR-CIRCUIT-SAT in a Suguru grid. We then design a physical Zero-Knowledge Proof (ZKP) protocol for Suguru. This ZKP protocol allows a prover to prove that he knows a solution of a Suguru grid to a verifier without leaking any information on the solution. To construct such a physical ZKP protocol, we only rely on a few physical cards and adapted encoding. For a Suguru grid with n cells, we only use $5n + 5$ cards. Moreover, we prove the three classical security properties of a ZKP: completeness, extractability, and zero-knowledge

Preliminary paper appears in: Léo Robert, Daiki Miyahara, Pascal Lafourcade, and Takaaki Mizuki. Physical zero-knowledge proof for suguru puzzle. In Stéphane Devismes and Neeraj Mittal, editors, *Stabilization, Safety, and Security of Distributed Systems - 22nd International Symposium, SSS 2020, Austin, TX, USA, November 18-21, 2020, Proceedings*, volume 12514 of *Lecture Notes in Computer Science*, pages 235–247. Springer, 2020

- Léo Robert, Daiki Miyahara, Pascal Lafourcade, and Takaaki Mizuki. Card-based ZKP for connectivity: Applications to nurikabe, hitori, and heyawake. *New Gener. Comput.*, 40(1):149–171, 2022

  Abstract: During the last years, several card-based Zero-Knowledge Proof (ZKP) protocols for Nikoli's puzzles have been designed. Although there are relatively simple card-based ZKP protocols for a number of puzzles, such as Sudoku and Kakuro, some puzzles face difficulties in designing simple protocols. For example, Slitherlink requires novel and elaborate techniques to construct a protocol. In this work, we focus on three Nikoli puzzles: Nurikabe, Hitori, and Heyawake. To date, no card-based ZKP protocol for these puzzles has been developed, partially because they have a relatively tricky rule that colored cells should form a connected area (namely, a polyomino); this rule, sometimes referred to as "Bundan-kin" (in Japanese), complicates the puzzles, as well as facilitating difficulties in designing card-based ZKP protocols.

  Preliminary paper appears in: Léo Robert, Daiki Miyahara, Pascal Lafourcade, and Takaaki Mizuki. Interactive physical ZKP for connectivity: Applications to nurikabe and hitori. In Liesbeth De Mol, Andreas Weiermann, Florin Manea, and David Fernández-Duque, editors, *Connecting with Computability - 17th Conference on Computability in Europe, CiE 2021, Virtual Event, Ghent, July 5-9, 2021, Proceedings*, volume 12813 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2021

- Pascal Lafourcade, Daiki Miyahara, Takaaki Mizuki, Léo Robert, Tatsuya Sasaki, and Hideaki Sone. How to construct physical zero-knowledge proofs for puzzles with a "single loop" condition. *Theor. Comput. Sci.*, 888:41–55, 2021

  Abstract: We propose a technique to construct physical Zero-Knowledge Proof (ZKP) protocols for puzzles that require a single loop draw feature. Our approach is based on the observation that a loop has only one hole and this property remains stable by some simple transformations. Using this trick, we can transform a simple big loop, which is visible to anyone, into the solution loop by using transformations that do not

disclose any information about the solution. We illustrate our technique by applying it to construct physical ZKP protocols for two Nikoli puzzles: Slitherlink and Masyu.

- Jannik Dreier, Jean-Guillaume Dumas, Pascal Lafourcade, and Léo Robert. Optimal Threshold Padlock Systems. *Journal of Computer Security*, pages 1–34, 2021

  Abstract: In 1968, Liu described the problem of securing documents in a shared secret project. In an example, at least six out of eleven participating scientists need to be present to open the lock securing the secret documents. Shamir proposed a mathematical solution to this physical problem in 1979, by designing an efficient k-out-of-n secret sharing scheme based on Lagrange's interpolation. Liu and Shamir also claimed that the minimal solution using physical locks is clearly impractical and exponential in the number of participants. In this work, we relax some implicit assumptions in their claim and propose an optimal physical solution to the problem of Liu that uses physical padlocks, but the number of padlocks is not greater than the number of participants. Then, we show that no device can do better for k-out-of-n threshold padlock systems as soon as $k \geq \sqrt{2n}$, which holds true in particular for Liu's example. More generally, we derive bounds required to implement any threshold system and prove a lower bound of $O(\log(n))$ padlocks for any threshold larger than 2. For instance we propose an optimal scheme reaching that bound for 2-out-of-n threshold systems and requiring less than $2\log_2(n)$ padlocks. We also discuss more complex access structures, a wrapping technique, and other sublinear realizations like an algorithm to generate 3-out-of-n systems with $2.5\sqrt{n}$ padlocks. Finally we give an algorithm building k-out-of-n threshold padlock systems with only $O\left(\log(n)^{k-1}\right)$ padlocks. Apart from the physical world, our results also show that it is possible to implement secret sharing over small fields.

- Xavier Bultel, Pascal Lafourcade, Charles Olivier-Anclin, and Léo Robert. Generic construction for identity-based proxy blind signature. In Esma Aïmeur, Maryline Laurent, Reda Yaich, Benoît Dupont, and Joaquín García-Alfaro, editors, *Foundations and Practice of Security - 14th International Symposium, FPS 2021, Paris, France, December 7-10, 2021, Revised Selected Papers*, volume 13291 of *Lecture Notes in Computer Science*, pages 34–52. Springer, 2021

  Abstract: Generic constructions of blind signature schemes have been studied since its appearance. Several constructions were made leading to generic blind signatures and achieving other properties such as identity-based blind signature and partially blind signature. We propose a generic construction for identity- based Proxy Blind Signature (IDPBS). This combination of properties has several applications in the real world, in particularly in e-voting or e-cash systems and it has never been achieved before with a generic construction. Our construction only requires two classical signatures schemes: a blind EUF-CMA blind signature and a SUF-CMA unique signature. The security of our generic identity-based proxy blind signature is proven under these assumptions

- Pascal Lafourcade, Léo Robert, and Demba Sow. Linear generalized elgamal encryp-

tion scheme. In Pierangela Samarati, Sabrina De Capitani di Vimercati, Mohammad S. Obaidat, and Jalel Ben-Othman, editors, *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications, ICETE 2020 - Volume 2: SECRYPT, Lieusaint, Paris, France, July 8-10, 2020*, pages 372–379. ScitePress, 2020

Abstract: ElGamal public key encryption scheme has been designed in the 80's. It is one of the first partial homomorphic encryption and one of the first IND-CPA probabilistic public key encryption scheme. A linear version has been recently proposed by Boneh et al. In this work, we present a linear encryption based on a generalized version of ElGamal encryption scheme. We prove that our scheme is IND-CPA secure under linear assumption. We design a generalized ElGamal scheme from the generalized linear. We also run an evaluation of performances of our scheme. We show that the decryption algorithm is slightly faster than the existing versions.

- Pascal Lafourcade, Léo Robert, and Demba Sow. Fast short and fast linear cramer-shoup. In Gabriela Nicolescu, Assia Tria, José M. Fernandez, Jean-Yves Marion, and Joaquín García-Alfaro, editors, *Foundations and Practice of Security - 13th International Symposium, FPS 2020, Montreal, QC, Canada, December 1-3, 2020, Revised Selected Papers*, volume 12637 of *Lecture Notes in Computer Science*, pages 121–136. Springer, 2020

  Abstract: A linear Cramer-Shoup encryption scheme version was proposed by Shacham in 2007. Short Cramer-Shoup encryption scheme was designed by Abdalla et al., in 2014. This scheme is a variant of the Cramer-Shoup encryption scheme that has a smaller size. They proved that it is an IND-PCA secure encryption under DDH and the collision-resistance assumptions. We design a faster version of Short Cramer-Shoup encryption scheme denoted Fast Short Cramer-Shoup encryption. We also propose a faster version of linear Cramer-Shoup encryption called Fast Linear Cramer-Shoup. We prove that the Fast Short Cramer-Shoup is IND-PCA secure under DDH and the collision-resistance assumptions. We also, show that our linear encryption is CCA secure under the Linear assumption. Finally we run an evaluation of performances of our schemes.

- Pascal Lafourcade, Léo Robert, and Demba Sow. Fast cramer-shoup cryptosystem. In Sabrina De Capitani di Vimercati and Pierangela Samarati, editors, *Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021, July 6-8, 2021*, pages 766–771. SCITEPRESS, 2021

  Abstract: Cramer-Shoup was the first practical adaptive CCA-secure public key encryption scheme. We propose a faster version of this encryption scheme, called Fast Cramer-Shoup. We show empirically and theoretically that our scheme is faster than three versions proposed by Cramer-Shoup in 1998. We observe an average gain of 60% for the decryption algorithm. We prove the IND-CCA2 security of our scheme. The proof only relies on intractability assumptions like DDH.

- Matthieu Journault, Pascal Lafourcade, Malika More, Rémy Poulain, and Léo Robert. How to teach the undecidability of malware detection problem and halting problem.

In Lynette Drevin, Suné von Solms, and Marianthi Theocharidou, editors, *Information Security Education. Information Security in Action - 13th IFIP WG 11.8 World Conference, WISE 13, Maribor, Slovenia, September 21-23, 2020, Proceedings*, volume 579 of *IFIP Advances in Information and Communication Technology*, pages 159–169. Springer, 2020

Abstract: Malware detection is a term that is often associated to Computer Science Security. The underlying main problem is called *Virus detection* and consists in answering the following question: Is there a program that can always decide if a program is a virus or not? On the other hand, the undecidability of some problems is an important notion in Computer Science : an undecidable problem is a problem for which no algorithm exists to solve it. We propose an activity that demonstrates that virus detection is an undecidable problem. Hence we prove that the answer to the above question is no. We follow the proof given by Cohen in his PhD in 1983. The proof is close to the proof given by Turing in 1936 of the undecidability of the Halting problem. We also give an activity to prove the undecidability of the Halting problem. These proofs allow us to introduce two important ways of proving theorems in Computer Science : proof by contradiction and proof by case disjunction. We propose a simple way to present these notions to students using a maze. Our activity is unplugged, *i.e.*, we use only a paper based model of computer, and is designed for high-school students. This is the reason why we use Scratch to write our ”programs“.

Chapter 1

Background on Provable Security

Contents

We aim at presenting, in a manner that is as detailed and as comprehensive as possible, all the basic components needed to grasp our work.

We consider two entities, denoted Alice and Bob, trying to communicate via a channel. An attacker, called Eve, could be listening or modifying the contents of the messages exchanged over this channel. This context is depicted in Figure 1.1.

In what follows we present several types of properties that cryptographic schemes might need to achieve, against which adversaries that security must be guaranteed, as well as the various tools that can help construction solutions that have the security we are aiming for.

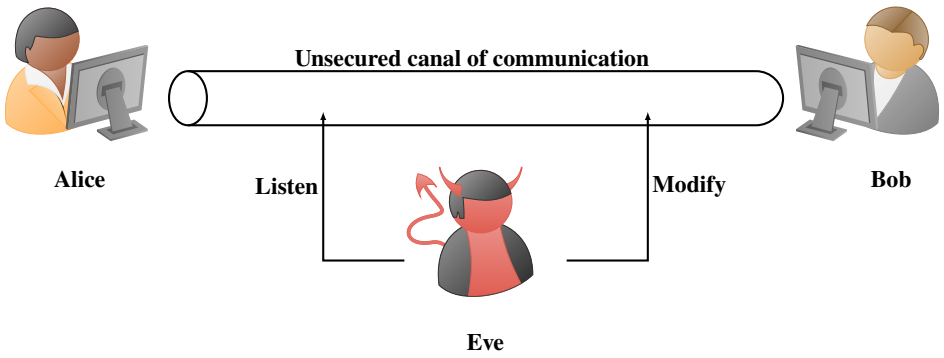


Figure 1.1 *The context of secure communications.*

1.1 Notations

We define the mathematical basis and notations we use through the manuscript.

Let g be a generator of a cyclic group $\mathbb{G}$ of prime order q. A user's Diffie-Hellman public key is an exponentiation of g to the private exponent k: $pk = g^k \mod p$ for a large prime p.

We end names of public keys in pk and private keys ending in k. For instance Rchpk is a ratchet public key with corresponding private key rchk.

A key generated by party P is denoted by ik_P while the public key is denoted ipk_P.

Stage-specific keys have stages as superscript *e.g.*,, $\mathsf{mk}^{1,1}$.

$DH(x, y) = x^y$ denotes the exponentiation of $x \in \mathbb{G}$ to a power $y \in \mathbb{Z}_q$.

$a||b$ is the concatenation of two bitstrings a and b.

$x \xleftarrow{\$} \mathcal{S}$ means that x is uniformly chosen at random from the set $\mathcal{S}$.

$|\mathcal{S}|$ is the cardinality of set $\mathcal{S}$. For a bitstring x, we denote by $|x|$ its length.

$\mathbb{P}[E]$ is the probability that event E occur.

The syntax of a primitive (or protocol) is characterized by tuple of (polynomial time) algorithms. For example, a public-key encryption scheme E is composed of $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with:

- $\underline{\mathsf{Gen}}$: takes as input a security parameter 1^λ to output a key pair $(\mathsf{sk}, \mathsf{pk})$;
- $\underline{\mathsf{Enc}}$: takes as input a public key pk and a message $m \in \mathcal{M}$ to output a ciphertext $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$;
- $\underline{\mathsf{Dec}}$: takes as input a private key sk and a ciphertext c to output a plaintext $m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$ or an error message $\perp$.

Moreover, we require that for every λ, every $(\mathsf{sk}, \mathsf{pk})$, and every message m, it holds that:

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$$

1.2 Security Properties

The proofs given in this work are computational, written for game-based definitions. The security properties are captured through *games*, or security experiments, played between an adversary and a challenger (which represents all the honest parties using the primitive/protocol). Both participants have access to the algorithms of the primitive/protocol but not necessarily their secret values [1]. Moreover, the adversary could have access to external

[1] As described by one the Kerckhoffs' principle (or Shannon maxim) which states that the only source of secret is from the keys and not the algorithms.

resources which is modelled by *oracles*. They represent the possible interactions with honest parties (*i.e.*, $\mathcal{A}$ could stop a message), with the primitive (*i.e.*, $\mathcal{A}$ could query an encryption of a given message).

We assume that the adversary (denoted $\mathcal{A}$) is a probabilistic polynomial time (PPT) algorithm. If the adversary calls oracle O with public parameter pparam on input x to output y then we write it as $y \leftarrow \mathcal{A}^{O(x)}(\mathsf{pparam})$. Note that oracle could be queried *adaptively*, in this case we write simply $y \leftarrow \mathcal{A}^{O(\cdot)}(\mathsf{pparam})$.

Negligible functions. The concept of negligible function is used to prove security in the sequence-of-games approach [Sho06] where two events can be considered equivalent as long as their corresponding probabilities that they occur are *close* enough (*i.e.*, negligible).

Definition 1 (Negligible function) *A function $f : \mathbb{N} \to \mathbb{R}$ is negligible if for all polynomial p, there exists an integer $n \in \mathbb{N}$ such that for all $x > n$:*

$$|f(x)| < \frac{1}{p(x)}$$

Note that for all integer $c > 1$ and for all positive polynomial p of degree ≥ 1, $\frac{1}{c^{p(x)}}$ is negligible.

Probability. We admit the following relations, given E and F two events:

$$\mathbb{P}[E] = \mathbb{P}[F] \cdot \mathbb{P}[E|F] + \mathbb{P}[\neg F] \cdot \mathbb{P}[E|\neg F]$$

$$\mathbb{P}[E \wedge F] = \mathbb{P}[E] \cdot \mathbb{P}[F] \text{ if } A \text{ and } B \text{ are independent}$$

Lemma 1 (Difference Lemma [Sho06]) *Let A, B, F be events defined in some probability distribution, and suppose that $A \wedge \neg F \iff B \wedge \neg F$. Then $|\mathbb{P}[A] - \mathbb{P}[B]| \leq \mathbb{P}[F]$*

This lemma is important when proving the security of cryptographic constructions, such as primitives, protocols, etc. Indeed, some proofs rely on a sequence of games, where the initial game is the initial experiment (capturing the security property) and the last game gives explicit analysis of the adversary to win the game (*e.g.*, a negligible probability if the scheme is secure).

Hopping from one game to the next is achieved by a slight modification, either of the scheme or the way the adversary interacts with it, but such that the attacker cannot tell that such a difference exists.

One way to prove that the hop from one game to another is sound (i.e., the adversary cannot distinguish between the two games) is by employing the Difference Lemma 1.

1.2.1 Reductions

Security proofs of new cryptosystems, in modern cryptography, consist of constructing a reduction $\mathfrak{R}$ (or multiple ones), which turns an efficient adversary $\mathcal{A}$ against the primitive (or protocol) into an other adversary $\mathfrak{R}^{\mathcal{A}}$ that solves a well-studied problem assumed to be hard (some of them are given below in 1.3). If we assume that the underlying hard problem is not solvable in polynomial time, then the protocol's security cannot be broken by the original adversary.

Basically, the reduction ensures that the cryptosystem is at least as hard to break as it is to solve a computational problem (assumed to be hard). Note that although we give only the general idea without further substantial details, the authors of [KM07] give analysis and critique on provable security results.

The quality of the reduction is given by how well the reduction behaves (in terms of running time and success probability) with respect to how fast/efficient the adversary was in the first place. Denote the running time of the reduction $\mathfrak{R}^{\mathcal{A}}$ by $\tau_{\mathfrak{R}^{\mathcal{A}}}(\lambda)$ and the success probability by $\epsilon_{\mathfrak{R}^{\mathcal{A}}}(\lambda)$ of $\mathfrak{R}^{\mathcal{A}}$ for some security parameter $\lambda \in \mathbb{N}$. If we also express them, respectively, for $\mathcal{A}$ as $\tau_{\mathcal{A}}(\lambda)$ and $\epsilon_{\mathcal{A}}(\lambda)$ then we have:

$$\tau_{\mathfrak{R}^{\mathcal{A}}}(\lambda)/\epsilon_{\mathfrak{R}^{\mathcal{A}}}(\lambda) = \ell(\lambda) \cdot \tau_{\mathcal{A}}(\lambda)/\epsilon_{\mathcal{A}}(\lambda) \ ,$$

where $\ell(\lambda)$ is the loss of the reduction with respect to the original adversary. For an *efficient* reduction, the loss is bounded by a polynomial. If $\ell(\lambda)$ is constant, then the reduction is *tight*.

1.2.2 Security Games

We now describe usual security games for public key encryption schemes, namely the *indistinguishability under chosen-plaintext attacks* (IND-CPA) and the *indistinguishability under chosen-ciphertext attacks* (IND-CCA). In each case, the game starts with the challenger running the key-generation algorithm to output a key pair $(\mathsf{sk}, \mathsf{pk})$. The public key is handed to the adversary and the challenger draws a random bit $b \in \{0, 1\}$. Then $\mathcal{A}$ outputs two messages m_0, m_1 (with the restriction m_0 and m_1 have the same length *i.e.*, $|m_0| = |m_1|$). The challenger sends to $\mathcal{A}$ the encryption of m_b and finally $\mathcal{A}$ outputs a bit b'.

We say that $\mathcal{A}$ wins if and only if $b' = b$. Depending on the game, the adversary has access to different oracles. In IND-CPA, the adversary has only access to the encryption oracle $O_{\mathcal{E}}$ (which is, in practice, natural since the public key is known). For IND-CCA, the adversary has also access to the decryption oracle $O_{\mathcal{D}}$ with the restriction that the challenge ciphertext cannot be an input. We differentiate the IND-CCA game in two depending on the call to the decryption oracle. If the oracle can be queried only before the challenge, then the game is called IND-CCA1 and if the oracle can also be called after the challenge then the game is denoted IND-CCA2.

Definition 2 (IND-CPA) *Let* $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme, we define experiment* IND-CPA *as:*

$\mathsf{Exp}_E^{\textit{IND-CPA}}(\lambda, \mathcal{A})$*:*
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$
$b \xleftarrow{\$} \{0, 1\}$
$(m_0, m_1) \leftarrow \mathcal{A}^{O_{\mathcal{E}}}(1^\lambda, \mathsf{pk})$
$c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b)$
$b' \leftarrow \mathcal{A}^{O_{\mathcal{E}}}(\mathsf{pk}, c)$
return $(b = b')$

We define the advantage of $\mathcal{A}$ against the IND-CPA security of E as:

$$\mathsf{Adv}_E^{\textit{IND-CPA}}(\lambda, \mathcal{A}) = \left| \mathbb{P}\left[1 \leftarrow \mathsf{Exp}_{E,\mathcal{A}}^{\textit{IND-CPA}}(\lambda)\right] - \frac{1}{2} \right|$$

We define the IND-CPA advantage against E as:

$$\mathsf{Adv}_E^{\textit{IND-CPA}}(\lambda) = \max_{\mathcal{A}} \left\{ \mathsf{Adv}_{E,\mathcal{A}}^{\textit{IND-CPA}}(\lambda) \right\}$$

We say that is E IND-CPA-secure if $\mathsf{Adv}_E^{\textit{IND-CPA}}(\lambda)$ *is negligible.*

Definition 3 (IND-CCA) *Let* $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme, we define experiment IND-CCA as:*

$\mathsf{Exp}_E^{\textit{IND-CCA}}(\lambda, \mathcal{A})$*:*
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$(m_0, m_1) \leftarrow \mathcal{A}^{O_\mathcal{E}, O_\mathcal{D}}(1^\lambda, \mathsf{pk})$
$c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b)$
$b' \leftarrow \mathcal{A}^{O_\mathcal{E}, O_\mathcal{D}}(\mathsf{pk}, c)$
return $(b = b')$

We define the advantage of $\mathcal{A}$ against the IND-CCA security of E as:

$$\mathsf{Adv}_E^{\textit{IND-CCA}}(\lambda, \mathcal{A}) = \left| \mathbb{P}\left[1 \leftarrow \mathsf{Exp}_{E,\mathcal{A}}^{\textit{IND-CCA}}(\lambda)\right] - \frac{1}{2} \right|$$

We define the IND-CCA advantage against E as:

$$\mathsf{Adv}_E^{\textit{IND-CCA}}(\lambda) = \max_{\mathcal{A}} \left\{ \mathsf{Adv}_{E,\mathcal{A}}^{\textit{IND-CCA}}(\lambda) \right\}$$

We say that E is IND-CCA-secure if $\mathsf{Adv}_E^{\textit{IND-CCA}}(\lambda)$ *is negligible.*

1.3 Cryptographic assumptions

We present some common assumptions made in modern cryptography.

1.3.1 DL, CDH, DDH

Let p be a prime number and $\lambda = |p|$ its bit-size. Let $\mathbb{G}$ be a cyclic group, of order p, generated by an element g.

Definition 4 (Discrete Logarithm (DL)) *Solving the discrete logarithm problem in* $(\mathbb{G}, p, g)$ *is assumed to be hard meaning that there exists a negligible function ϵ such that for all PPT $\mathcal{A}$:*

$$\mathbb{P}\left[x \xleftarrow{\$} \mathbb{Z}_p; x' \leftarrow \mathcal{A}(g^x) : x = x' \right] = \epsilon(\lambda)$$

Definition 5 (Computational Diffie-Hellman (CDH)) *Solving the Computational Diffie-Hellman problem in* $(\mathbb{G}, p, g)$ *is assumed to be hard meaning that there exists a negligible function ϵ such that for all PPT $\mathcal{A}$:*

$$\mathbb{P}\left[(x, y) \xleftarrow{\$} (\mathbb{Z}_p)^2; z \leftarrow \mathcal{A}(g^x, g^y) : z = g^{x \cdot y} \right] = \epsilon(\lambda)$$

Definition 6 (Decisional Diffie-Hellman (DDH)) *Solving the Decisional Diffie-Hellman problem in* $(\mathbb{G}, p, g)$ *is assumed to be hard meaning that there exists a negligible function* ϵ *such that for all PPT* $\mathcal{A}$*:*

$$\left| \mathbb{P}\left[\begin{array}{l} (x, y, z_0) \xleftarrow{\$} (\mathbb{Z}_p)^3; z_1 = x \cdot y; \\ b \xleftarrow{\$} \{0,1\}; b' \leftarrow \mathcal{A}(g^x, g^y, g^{z_b}) \end{array} : b = b' \right] - \frac{1}{2} \right| = \epsilon(\lambda)$$

1.3.2 Gap Diffie-Hellman

The Gap Diffie-Hellman (GDH) problem introduced in [OP01] by Okamoto and Pointcheval *combines* the CDH problem and the DDH. Indeed, the GDH problem resembles the CDH problem while allowing the adversary to have access to the DDH oracle. This somewhat hybrid notion is used for instance for schemes that derive keys from some secret (which is a DH value) by using some Key Derivation Function (KDF, described in 1.6). In this setting, the challenger needs to return valid session keys without knowing the discrete logarithm of all the key-shares. The DDH oracle guarantees consistency of the derived keys. The oDDH oracle takes three inputs, g^a, g^b and C from some group $\mathbb{G}$ and returns 1 if and only if $g^{ab} = C$.

Definition 7 (Gap Diffie-Hellman [OP01]) *Solving the Gap Diffie-Hellman problem in* $(\mathbb{G}, p, g)$ *is assumed to be hard meaning that there exists a negligible function* ϵ *such that for all* $\mathcal{A} \in \textsc{poly}(\lambda)$*:*

$$\mathbb{P}\left[(x, y) \xleftarrow{\$} (\mathbb{Z}_p)^2; Z \leftarrow \mathcal{A}^{\mathsf{oDDH}(g^a, g^b, C)}(g^x, g^y) : Z = g^{x \cdot y} \right] = \epsilon(\lambda)$$

1.4 Authenticated Encryption with Associated Data (AEAD)

AEAD stands for Authenticated Encryption with Associated Data, first introduced in [Rog02]. Authenticated encryption provides confidentiality for the encrypted data, and authenticity and integrity for both the encrypted plaintext and the AD. These guarantees can be expressed separately, or monolithically in a single definition. We take the latter approach, following the definition of Hoang *et al.* [HKR15].

Definition 8 (AEAD-scheme) *An authenticated encryption scheme with associated data (AEAD-scheme) is composed of a tuple of three algorithms* $\mathtt{AEAD} = (\mathcal{K}, \mathtt{AEAD.Enc}, \mathtt{AEAD.Dec})$*. Associated to* $\mathtt{AEAD}$ *are sets* $\mathtt{Nonce} = \{0,1\}^\lambda$ *(with* λ *the security parameter),* $\mathtt{Message} \subseteq \{0,1\}^*$ *(satisfying* $M \in \mathtt{Message} \implies M' \in \mathtt{Message}$ *for any* M' *of same length of* M*) and finally* $\mathtt{Header} \subseteq \{0,1\}^*$*.*

The key space $\mathcal{K}$ *is a nonempty set of strings. The encryption algorithm* $\mathtt{AEAD.Enc}$ *is a deterministic algorithm with inputs* $K \in \mathcal{K}, N \in \mathtt{Nonce}, H \in \mathtt{Header}, M \in \mathtt{Message}$*, and outputs* $C = \mathtt{AEAD.Enc}_K(N, H, M)$*. The decryption algorithm* $\mathtt{AEAD.Dec}$ *is a deterministic algorithm with inputs* $K \in \mathcal{K}, N \in \mathtt{Nonce}, H \in \mathtt{Header}, C \in \{0,1\}^*$*, and outputs a string in* $\mathtt{Message}$ *or a symbol* $\perp$ *subject to the restriction that* $\mathtt{AEAD.Dec}_K^{N,H}(\mathtt{Enc}_K(N, H, M)) = M$ *for all* K, N, H, M*.*

The AEAD security property is defined in terms of two oracles oAEnc and oADec, which both depend on the same secret bit b, the same key K and the same list of existing

ciphertexts $\mathcal{L}_{\mathsf{enc}}$. In the security game the adversary aims to find the value of the hidden bit b.

$\mathsf{oAEnc}_b(N, H, M)$	$\mathsf{oADec}_b(N, H, c)$

if $b = 0$
  $c \leftarrow \mathtt{Enc}_K(N, H, M)$
else
  $c \leftarrow \{0,1\}^{\lvert\mathtt{Enc}_K(N,H,M)\rvert}$
$\mathcal{L}_{\mathsf{enc}} \leftarrow \mathcal{L}_{\mathsf{enc}} \cup \{c\}$
return c
	if $c \notin \mathcal{L}_{\mathsf{enc}}$
	  if $b = 0$
	  $m \leftarrow \mathtt{Dec}_K(N, H, C)$
	  else
	  $m \leftarrow \perp$
	else
	  $m \leftarrow \perp$
	return m

Figure 1.2 *The AEAD security game*

$\mathsf{Exp}^{\mathsf{AEAD}}_{\Pi}(\lambda, \mathcal{A})$:
$K \leftarrow_{\$} \mathcal{K}$ $b \leftarrow \{0,1\}$ $b^* \leftarrow \mathcal{A}^{\mathsf{oAEnc}(\cdot,\cdot,\cdot),\mathsf{oADec}(\cdot,\cdot,\cdot)}()$
$\mathcal{A}$ wins iff.: $b = b^*$

Figure 1.3 *The experiment* $\mathsf{Exp}^{\mathsf{AEAD}}_{\Pi}(\lambda, \mathcal{A})$.

We define the advantage of an adversary $\mathcal{A}$ in the $\mathsf{Exp}^{\mathsf{AEAD}}_{\Pi}(\lambda, \mathcal{A})$ security game by:

$$\mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{Enc},\mathsf{Dec}}(\mathcal{A}) = \left|\mathbb{P}[\mathcal{A} \text{ wins}] - \frac{1}{2}\right|.$$

Definition 9 (AEAD) *An authenticated encryption with associated data scheme* `AEAD` *is* (q, ϵ)*-AEAD-secure if for any PPT adversary making at most* q *queries to the* oAEnc *oracle it holds that:*

$$\mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{Enc},\mathsf{Dec}}(\mathcal{A}) \leq \epsilon.$$

We say an AEAD scheme is q*-AEAD secure if* ϵ *is negligible (as a function of the security parameter which is removed for simplicity) for any PPT adversary* $\mathcal{A}$*.*

1.5 Hash Functions

A cryptographic hash function is a function H that takes an input of arbitrary size and compresses it to a shorter output. The size of the output depends on the security parameter λ. By noting $\ell(\lambda)$, or simply ℓ, the size of the output, we have:

$$H : \{0,1\}^* \rightarrow \{0,1\}^{\ell}$$

There are many properties we might want hash functions to have, but there is one specifically we are interested in: the collision resistance. It says that it should be hard to find two different inputs outputting the same value. Note that the size of the output is smaller than hash function's domain so there are always collisions, but we want to avoid them as much as possible.

Definition 10 (Collision-Resistance) *A hash function $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$ is collision-resistant if for all adversary $\mathcal{A}$ there exists a negligible function ϵ such that:*

$$\mathbb{P}[(x, x') \leftarrow \mathcal{A}(H) : H(x) = H(x')] \leq \epsilon$$

Random Oracle Model (ROM). An ideal hash function with output considered as random and uniformly distributed in its output space is called a *random oracle.*. When a security proof includes a hash function assumed to be a random oracle, we say that the proof is in the random oracle model [BR93c].

Notice that any security proof provided in the ROM by idealizing at least one of the employed hash functions will only hold in the real world (with the real hash functions) as long as the latter behaves identically (from an adversarial standpoint) to a random oracle.

There exist some security gaps between these two worlds, as discussed in [Nie02]. Random oracles are helpful for programming security reductions where the challenger may want to choose specific outputs to return (*e.g.*, complete a simulation, force the adversary to launch an attack); this is possible as long as outputs are uniformly distributed (which is not the case for hash functions). For this reason, proofs in the ROM are easier to construct, and often require less security assumptions than those outside this model.

1.6 Hash Key Derivation Function (HKDF)

Some protocols with key evolution (*e.g.*, Signal) use Hash Key Derivation Function (HKDF) algorithm [Kra10] which has two main components: Extract and Expand. The Extract operation has an *input key material* and an optional *salt*. The Expand operation takes a *secret* (usually the output of a previous Extract), a *label* (publicly-known string), and an *input*. We first describe the more generic notion of a KDF and then the implementation based on HMAC (which is the actual HKDF scheme).

Definition 11 (secure-KDF [Kra10]) *We say that KDF is (t, q, ϵ)-secure with respect to a source keying material Σ if no adversary $\mathcal{A}$ running in time t and making at most q queries can win the experiment of Figure 1.4 with probability larger than $\frac{1}{2} + \epsilon$.*

Note that the adversary has access to both α and r.

Generic KDF. An extract-the-expand key derivation function KDF has two modules:

- a randomness extractor XTR: it produces "close-to-random" (in the statistical or computational sense) outputs; it may be deterministic or keyed via a salt value.
- a variable length output PRF where the key is the output from XTR (which we denote PRK).

There are two steps to produce the L-bit key material KM:

1. $PRK = XTR(\mathsf{salt}, \Sigma)$
2. $KM = PRF(PRK, \mathsf{info}, L)$

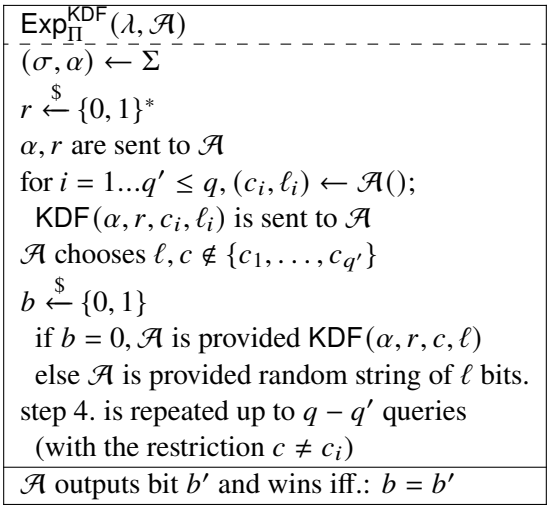


Figure 1.4 *The experiment* $\mathsf{Exp}^{\mathsf{KDF}}_{\Pi}(\lambda, \mathcal{A})$.

HMAC based KDF. We give the instantiation of the KDF solely based on HMAC which is called HKDF. In this scheme, HMAC is employed both for extraction and for expansion, which work as follows:

$$\mathsf{HKDF}(\mathsf{salt}, \Sigma, \mathsf{info}, L) = K(1)||\ldots||K(t)$$

where:

$$\begin{aligned} PRK &= \mathsf{HMAC}(\mathsf{salt}, \Sigma) \\ K(1) &= \mathsf{HMAC}(PRK, \mathsf{info}||0) \\ K(i+1) &= \mathsf{HMAC}(PRK, K(i)||\mathsf{info}||i) \end{aligned}$$

with $t = \lceil \frac{L}{k} \rceil$ and the last value $K(t)$ being truncated to its first $d = L \bmod k$ bits. Notice that length of the HMAC output is equal to the input key length (so the scheme is well defined).

1.7 Signature Schemes

Authentication is achieved, in asymmetric setting, by use of signature schemes [2]. While the signature literature is extremely vast, we only give here the formal details needed for the following sections.

Definition 12 (Signature [Kat10]) *A* signature scheme *is composed of three probabilistic polynomial-time algorithms:*

`Setup`: *a key-generation algorithm which takes in input the security parameter and outputs a pair of keys (*pk,sk*).*

`Sign`: *the signing algorithm which takes in input a secret key* sk *and a message m to produce a signature* σ*. We write this as* $\mathtt{Sign}_{\mathsf{sk}}(m)$.

[2] the corresponding notion in symmetric could be given by the MAC

`Verify`: *the verification algorithm which takes as input a public key* pk*, a message m, and a signature* σ *to return* 1 *(success) or* 0 *(failure). We write this as* $\texttt{Verify}_{\mathsf{pk}}(m, \sigma)$.

The security property guaranteed by most digital signature schemes, existential unforgeability against adaptively-chosen message attacks (EUF-CMA) , is expressed in terms of the stateful signing oracle oSign described below:

oSign$(\cdot)$: The signing oracle assumes the existence of a previously initialized list $\mathcal{L}_{\mathsf{sign}}$ of messages and of a signature public-private key pair (pk,sk) (which the challenger will instantiate as shown in Figure 1.5). On input an adversarially chosen message m, the oracle runs $\sigma \leftarrow \texttt{Sign}_{\mathsf{sk}}(m)$, updates $\mathcal{L}_{\mathsf{sign}} \leftarrow \mathcal{L}_{\mathsf{sign}} \cup m$, and outputs the signature σ to the adversary.

The EUF-CMA security experiment is depicted in Figure 1.5.

$\mathsf{Exp}^{\mathsf{EUF-CMA}}_{\Pi}(\lambda, \mathcal{A})$
$1.(\mathsf{sk}, \mathsf{pk}) \leftarrow \texttt{aSetup}(1^{\lambda})$
$\mathcal{L}_{\mathsf{sign}} \leftarrow \emptyset$
$2.(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{oSign}(\cdot)}(\mathsf{pk})$
$\mathcal{A}$ wins iff.:
$\texttt{Verify}_{\mathsf{pk}}(m^*, \sigma^*) = 1$
AND $m^* \notin \mathcal{L}_{\mathsf{sign}}$

Figure 1.5 *The experiment* $\mathsf{Exp}^{\mathsf{EUF-CMA}}_{\Pi}(\lambda, \mathcal{A})$.

We define the *advantage* of an adversary $\mathcal{A}$ against the $\mathsf{Exp}^{\mathsf{EUF-CMA}}_{\Pi}(\lambda, \mathcal{A})$ experiment as its success probability, taken over all the random coins of the challenger and the adversary; we denote this advantage as $\mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathsf{Sign}}(\mathcal{A})$. We generally parametrize the notion by the number of queries $q = |\mathcal{L}_{\mathsf{sign}}|$ the adversary makes to its signing oracle and by the upper bound ϵ on its advantage, as follows:

Definition 13 (EUF-CMA) *A signature scheme (*`Setup`*,* `Sign`*,* `Verify`*) is* $(q, \epsilon) - \mathsf{EUF} - \mathsf{CMA}$*−secure if for any PPT adversary making at most q queries to the* oSign *oracle it holds that:*

$$\mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathsf{Sign}}(\mathcal{A}) \leq \epsilon.$$

We say a signature scheme is $q - \mathsf{EUF} - \mathsf{CMA}$*−secure if* ϵ *is negligible (as a function of the security parameter) for any PPT adversary* $\mathcal{A}$.

In the following, we assume that all signature schemes involve hashing, and omit the hashing in the notation, *i.e.*, $\texttt{SIGN}_{\mathsf{sk}}(m) := \texttt{Sign}_{\mathsf{sk}}(H(m))$ for a hash function H.

CHAPTER 2

IMPROVING THE POST-COMPROMISE SECURITY OF SIGNAL

Communication between peers are a cornerstone in our today's societies. The widespread and popular usage of cryptography is mandatory (see [Rog16] for detailed arguments) especially after the revelations of Edward Snowden about mass surveillance. Ensuring security and privacy for day-to-day communication services has naturally arisen since. The research community, but also the industry and even regular users, became eager to develop and communicate by using secure messaging protocols.

We propose two variants of Signal, that we call MARSHAL and SAMURAI. Both aim at improving the Post-Compromise Security of Signal. Although MARSHAL and SAMURAI have the same level of security, SAMURAI outperforms MARSHAL in terms of memory management (and thus practicality).

Contents

2.1 Introduction

Numerous asynchronous messaging protocols have been designed, for instance Signal [MP16a], OTR [BGB04], Matrix [mat19], Wire [Gmb21], ART [CCG+18] and MLS [BBR+22].

The main goal of those protocols is to ensure the confidentiality and authenticity of the exchanged messages with respect to a PitM[1] while allowing the communicating partners to

[1]Person-in-the-Middle is a politically-correct version of Man-in-the-Middle.

be online at different time. That is, if Alice wants to talk to Bob, both are not needed to be online simultaneously in order to communicate.

Depending on the protocol, other security properties are featured. The case of Signal is interesting because of two of its properties (depicted below): Perfect Forward Secrecy (PFS) et Post-Compromise Security (PCS).

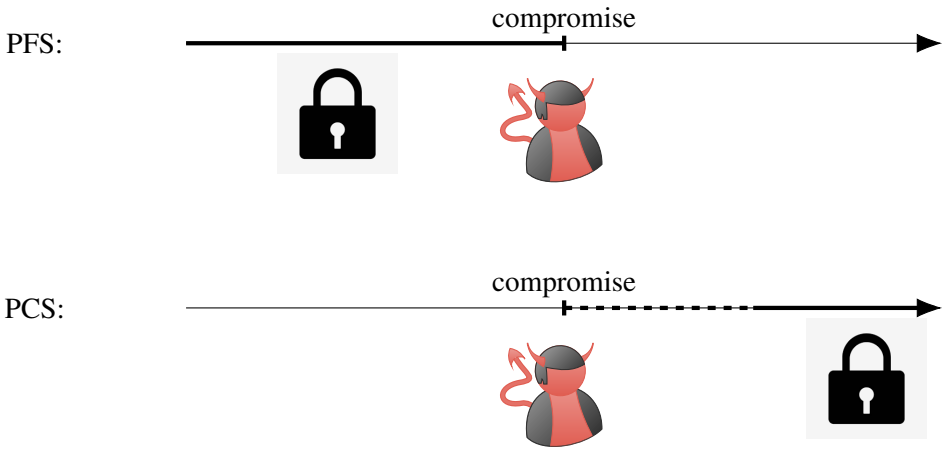


Figure 2.1 *PFS and PCS illustrations. The dashed lined indicates a compromised state while the thick parts represent a secure state.*

Signal guarantees *Perfect Forward Secrecy* (PFS), which ensures that compromising the long-term keys will not endanger the confidentiality of past communications. In addition, however, Signal also provides an orthogonal property called *Post-Compromise Security* (PCS), which is much more rare than PFS. The PCS notion, formalized in [CCG16], guarantees security even after a compromise. Depending on which keys are revealed, an attacker has access to some part of the communication until a given moment where the protocol *heals*. As the honest parties add randomness to the protocol, the attacker can no longer use the information it previously had to breach the security of the channel. This powerful yet conditional property is first analysed (in Signal context) in [CGCD+17].

The design of Signal is dedicated to long-term communication between peers. Hence, the sessions created are meant to last for long time. The PCS becomes a crucial property in this context: if a communication partner is compromised (thus loosing the confidentiality of the communication) then the PCS property guarantees that at some point in the future, the communication will heal from prior confidentiality breaches. In practice, a fast healing is preferable as the revealed part of the communication should be as short as possible.

In a nutshell, Signal can be split into 4 phases:

- **Initial key exchange**: non-interactive key-exchange (X3DH) through the help of a server;
- **Symmetric ratchet**: derivation of message keys through a KDF (see 1.6) for a party sending multiple messages without a reply from the partner; this step ensures PFS;
- **Asymmetric ratchet**: when the speaker changes, a new Diffie-Hellman is injected in the derivation of keys; this step ensures PCS;

- **Authenticated Encryption**: each message is AEAD (see 1.4) encrypted using a fresh message key, with an associated (authenticated) data consisting of auxiliary information that is necessary for the message's receiver to ratchet its keys.

In our work, we focus on key-updates, up to the transmission of messages; however, we focus less on the security of the X3DH, detailed in Section 2.2). Some works as [BFG+22, BFG+20, HKKP21] study the handshake of Signal (in the post-quantum paradigm). Other works focus on Signal for group communication [CPZ20] and multi-device [CDDF20, WBPE21]. The work of [KBB17] use ProVerif and CryptoVerif to analyze a modified version of the Signal protocol while their main goal is to provide a methodology for automated verification toward secure messaging protocols in general. Their work is compatible with a potential security analysis of our protocol; however, in this manuscript we take the complementary approach of providing a computational security analysis of Signal, as the one provided by [CGCD+17].

2.1.1 Contributions

Signal's PCS guarantee is limited by two main factors: the lack of persistent authentication (noticed by Blazy *et al.* [BBB+19]) and the frequency of asymmetric ratchets, which is our key motivation. Our goal is to design a PCS-improved protocol (actually there will be two protocols) sticking as close as possible to Signal. As we use Signal as the backbone of our design, it is more straightforward to compare the three protocols.

For this, we design two protocols, MARSHAL [BFJ+22] (Messaging with Asynchronous Ratchets and Signatures for faster HeALing) and SAMURAI (Signal-like Asynchronous Messaging with Message-loss resilience, Ultimate healing and Robustness against Active Impersonations) adding a faster healing and persistent authentication. Both protocols guarantee the same level of healing (*i.e.*, they have the same security given a compromise), but SAMURAI is more efficient in terms of complexity and memory.

These strong properties come at a cost. Apart from registering an additional DH element and having to perform DH computations at each stage, MARSHAL adds to the complexity of Signal in two ways: (1) requiring the transmission of a number of DH elements that is linear in the maximal depth of the chain; (2) using signatures to transmit the encrypted messages and stage metadata. The former allows us to provide message-loss resilience: if this is not needed, metadata size can be reduced. The second source of complexity, the signatures, serves a double purpose: they help preserve AKE security, and they restrict an adversary's ability to impersonate parties upon corruption.

In order to facilitate the understanding of our contributions, we give in Figure 2.2, a toy-example of a Signal conversation between an *initiator* Alice and the *responder* Bob. Each message comes at a protocol *stage*[2], denoted by (x, y). The y value increases when the speaker changes (Alice starts at $y = 1$, then y turns to 2 when Bob speaks, etc.). The x value increases with each new message from the same speaker, and is reset to 1 for each new value of y. Each stage (x, y) is associated with a message key $\mathsf{mk}^{x,y}$, used to encrypt and authenticate that stage's message. To evolve from a key $\mathsf{mk}^{x,y}$ to $\mathsf{mk}^{x+1,y}$ (next message,

[2]We use a different notation of stages from [CGCD+17].

same speaker), the two peers use a key-derivation function (KDF) with no further freshness. This is called a *symmetric ratchet*, denoted by [S] in Figure 2.2. To update a key $\mathsf{mk}^{x,y}$ to $\mathsf{mk}^{1,y+1}$ (new speaker), a DH share is used as freshness into the KDF. This is called an *asymmetric ratchet*, denoted by [A].

Sender	**Key(s)**	**AD**	**Message**	MARSHAL/SAMURAI	**Signal**
Alice	$\mathsf{mk}^{1,1}$:	$(1, \mathsf{Rchpk}_A^1)$	Hi Bob	✓	✓
	[S] $\mathsf{mk}^{2,1}$	$(2, \mathsf{Rchpk}_A^1)$	How are you ?	×	×
	[S] $\mathsf{mk}^{3,1}$-$\mathsf{mk}^{17,1}$	$(3, \mathsf{Rchpk}_A^1)$-$(17, \mathsf{Rchpk}_A^1)$	(... *15 messages*)	✓	×
	[S] $\mathsf{mk}^{18,1}$	$(18, \mathsf{Rchpk}_A^1)$	Cinema tonight ?	✓	×
Bob :	[A] $\mathsf{mk}^{1,2}$	$(1, \mathsf{Rchpk}_B^2)$	Hi Alice	✓	×
	[S] $\mathsf{mk}^{2,2}$	$(2, \mathsf{Rchpk}_B^2)$	I'm good, thanks	✓	×
	[S] $\mathsf{mk}^{3,2}$-$\mathsf{mk}^{12,2}$	$(3, \mathsf{Rchpk}_B^2)$-$(12, \mathsf{Rchpk}_B^2)$	(... *10 messages*)	✓	×
Alice :	[A] $\mathsf{mk}^{1,3}$	$(1, \mathsf{Rchpk}_A^3)$	Great	✓	✓

Figure 2.2 *Toy example for Signal and* MARSHAL *and* SAMURAI. *Messages are encrypted with the keys in the second column (indexed by the stage) and have the associated data (AD) in the third column. The labels [A] and [S] indicate asymmetric and symmetric ratcheting respectively. The security (✓) and insecurity (×) of messages is given, assuming Alice is compromised at message 2. Italics show that several messages are sent in the same chain.*

2.1.2 Related Work

Ratcheted key-exchange (RKE) was introduced as a unidirectional, single-move primitive by Bellare *et al.* [BSJ+17], who used it to define and instantiate ratcheted encryption. This security model was later extended by work such as [PR18, JS18] to treat double ratchets, but also more generic RKE. A crucial difference between generic RKE and our work is that we focus on the full message transmission process, as in the case of [CGCD+17, BBB+19]. The work in [CDV21] proposes generic constructions for ratcheting algorithms.

The work of Alwen *et al.* [ACD19] provides a complete security model for protocols like Signal, which also handles out-of-order messages (called in [ACD19] an *immediate decryption*). Alwen *et al.* view asynchronous messaging protocols as a composition of three parts: a hash function that generates pseudorandom output (PRF-PRNG), a primitive called forward-secure AEAD (FS-AEAD) which captures symmetric ratchets, and continuous key-agreement (CKA) which captures asymmetric ratchets. While this work does capture Signal and allows for modular security proofs, it is not so well suited to the analysis of our protocol, for three main reasons. First, MARSHAL (and SAMURAI) do not employ any symmetric ratcheting; second, we want to capture the properties of the actual message transmission; third, we do not use AEAD solely, but rather combine it with a public-key authentication mechanism. This would minimally indicate a need to modify the FS-AEAD primitive. We therefore prefer a security model that is less modular, but comes closer to the protocol (as in [BBB+19]). We have adapted one of the properties they consider to our security model, namely that of message-loss resilience.

The works of [JMM19, DV19] provide efficient instantiations of bidirectional ratcheted

key-exchange by using relatively inexpensive primitives (unlike previous work such as *e.g.*, that of Poettering and Rössler [PR18]). However, these protocols are different and do not follow the structure of Signal. In addition, features such as out-of-order messages are not included, and some of these constructions *require* the parties to receive each message. Starting from Signal's structure, we preserve properties such as out-of-order messages, and have stronger healing by persistent authentication and more frequent asymmetric ratchets.

Our work comes closest to the SAID protocol of Blazy *et al.* [BBB+19], whose notion of persistent authentication prevents hijacking attacks. SAID therefore authenticates each ratchet by using identity keys. As long as the identity keys are safely stored, session-hijacking cannot happen because the adversary cannot convince Bob he is ratcheting with the correct person. While we also ensure persistent authentication, our work uses the backbone of Signal and its security assumptions: public-key cryptography and a semi-trusted middle server. By contrast, Blazy *et al.* constructed their protocol in the paradigm of identity-based cryptography.

An interesting work, but which is orthogonal to ours was presented at CCS '19 by Chase et al. [CDGM19]. They focus on the long-term keys of two parties, and present a way of updating those in case access is lost to a user's current account. We did not consider updates to long-term key in this paper; instead we focus on the actual session and message keys.

Outline. We start by describing the Signal protocol in Section 2.2 to better explain our first variant MARSHAL in Section 2.3. Then we present in Section 2.4, our second variant SAMURAI, which has the same security as MARSHAL but with better efficiency. Finally, we propose implementations of those protocols to evaluate their practical costs in Section 2.5 before concluding in Section 2.6.

2.2 Description of Signal

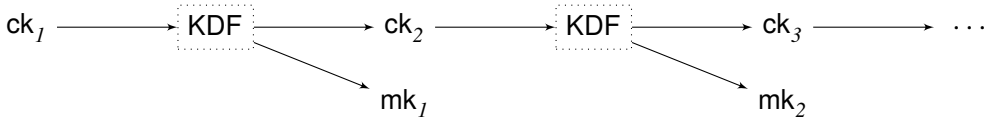
Before completely detailing Signal protocol, we give first a generic description of its core operating. Indeed, Signal is based on the double ratchet algorithm proposed in [MP16b]. After presenting it, we give the detailed description of Signal.

2.2.1 Double ratchet algorithm

The term *ratchet* refers to the fact that a given key is derived from a previous one. The output key is indistinguishable from a random value (as a PRF), and it is hard to inverse the derivation (*i.e.*, no one can recover the input by knowing only the output). There are two types of ratcheting, symmetric and asymmetric. In both cases, the keys are chained through a KDF (see 1.6).

Symmetric ratchet. The keys are chained through a KDF with no additional value added during the process. We denote the input keys and output keys with the term *chain key* ck. In the case of Signal, there is an other output used to encrypt the message, namely the message key mk.

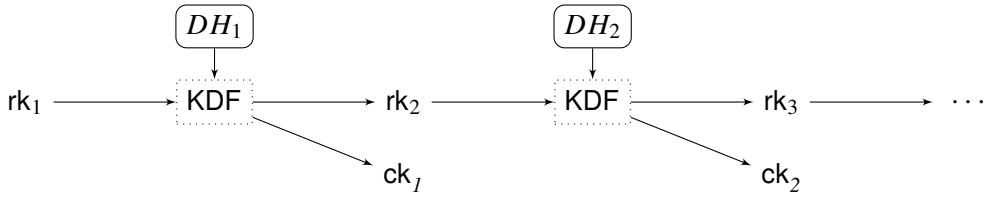
The symmetric ratchet can be resumed as:



Notice that once a chain key ck_x (for a given x) is known then all the subsequent keys $\mathsf{ck}_{x'>x}$ and $\mathsf{mk}_{x'\geq x}$ can be efficiently computed (induced by the definition of KDF). However the inverse is not true meaning that there is no efficient way of computing ck_{x-1} from the knowledge of ck_x (which captures the PFS property).

Asymmetric ratchet. To prevent an attacker to recover the full chain when compromising a chain key, random values are added in each derivation. Now, the output chain key is not fully determined by its input chain key. The freshness added in each state of the chain is a Diffie-Hellman value (denoted DH) ensuring that both partner can derive the keys. We call the input and output *root keys* rk to differentiate with the symmetric ratcheting. Note that there is also an additional output key which is simply a chain key ck.

The asymmetric ratchet is illustrated as follows:



The key schedule of Signal is based on those two algorithms which are composed together to form the double ratchet algorithm. We now proceed to the detailed description of this protocol.

2.2.2 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 Figure 2.3, 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 ipk_P, a medium-term key prepk_P signed with ik_P, and optional ephemeral *public* keys 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 [MP16a] to generate an initial shared secret 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

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

Figure 2.3 *A Signal protocol session between initiator Alice and responder Bob.*

(KDF_r), outputting the root key 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 Epk_A, the ratchet key 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 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 Rchpk^y and the stage index[3] $x + 1$. 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 Rchpk^y. When Bob comes online, he chooses a new ratchet key rchk^{y+1}, and the public key 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[4] rk_y and the Diffie-Hellman product $(\mathsf{Rchpk}^y)^{\mathsf{rchk}^{y+1}}$ via KDF_r. Then, the chain and message keys are computed as described in the previous item. From that point onward, 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 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 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.

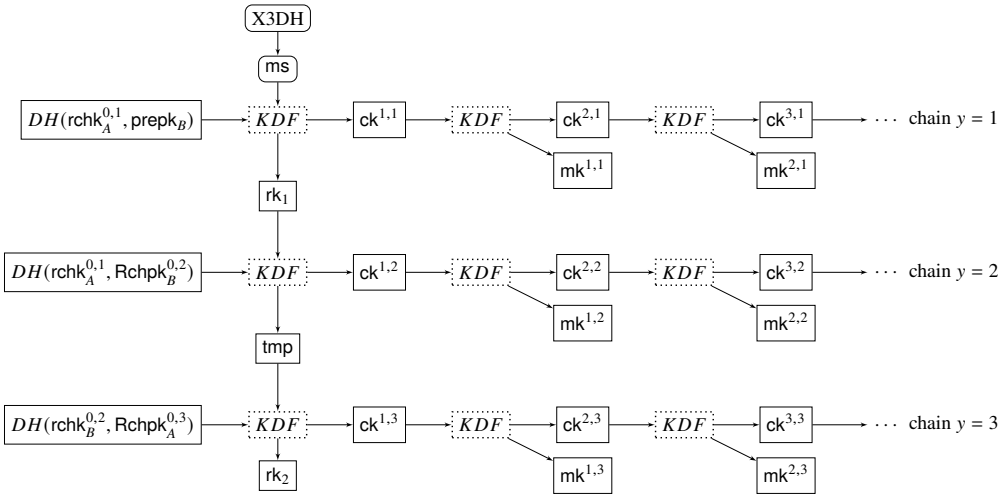


Figure 2.4 *The key schedule of Signal. Each stage (x, y) has its x-coordinate corresponding to a message (horizontal moves) inside a chain (vertical moves) for y-coordinate.*

[3] 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.

[4] 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.

2.3 MARSHAL

The protocol we propose, Messaging with Asynchronous Ratchets and Signatures for faster HeALing (MARSHAL), runs –like Signal– in several stages: registration, session setup, and communication. We describe in Figure 2.5 the session-setup and communication phases of MARSHAL. As a novelty, MARSHAL requires two types of ratchet keys: *same-user* ratchet keys, and *cross-user* ratchet keys. Same-user ratchet keys are indexed by stage, and generated whenever a new message is sent: for instance $\mathsf{Rchpk}^{2,1}$ denotes the ratchet public key at stage $(2, 1)$ (the second message sent in the first message chain). Cross-user ratchet keys are only generated at the beginning of a chain of messages and indexed only by the y-component of the stage (called a *chain index*). We denote by T_i the public key generated during the i-th message chain and by T_0 an initial public key registered by the session's responder.

While stages are indexed as (x, y) with $x, y \geq 1$, special indexes $x = 0$ and $y = 0$ denote special ratchet keys used for initialization. The first same-user ratchet key $\mathsf{Rchpk}^{0,1}$ is only used to compute the master secret of a session. Additionally, a ratchet key T_0 is registered by each user. The initiator of a session uses its correspondent's initial ratchet key during the first chain of communication ($y = 1$). Note that this method of ratcheting uniquely associates stages and chain indexes to the party generating them.

2.3.1 Registration

To use MARSHAL, each party P must first *register*, by generating private keys and uploading the corresponding public keys to the server: a long-term identity key ik_P; a medium-term prekey prek_P, and a signature on that key (generated with the identity key ik_P); multiple ephemeral one-time-use prekeys ephpk_P; multiple medium-term stage keys T_0. The last of these keys is a cross-user ratchet key (see above): a novelty with respect to Signal, which will help Alice asymmetric-ratchet in the first chain of messages, when she has not yet had a message (and therefore a ratcheting key) from Bob. In addition to these keys users will also generate and subsequently use a pair of long-term signature keys (sk_P, pk_P). These keys will not be registered on the server, but rather included in the associated data in each partner's first respective chain of messages.

2.3.2 Session Setup

Whenever Alice A wants to contact Bob B, she runs a protocol similar to that of Signal and [MP16a], with some small tweaks.

The master secret. To initiate a session with B, Alice queries the server for Bob's following values: the identity key ipk_B, a signed prekey prepk_B, a one-time prekey ephpk_B (if available), and a medium-term stage key T_B^0, denoted in short T_0. Having received those keys, A generates its own ephemeral key ek_A. The master secret ms is a concatenation of DH values, as computed in Signal.

First keys. Alice randomly generates a same-user ratchet key-pair $(\mathsf{rchk}^{0,1}, \mathsf{Rchpk}^{0,1})$. She computes a DH of her ratchet key and prepk_B, and the result is fed to a key derivation

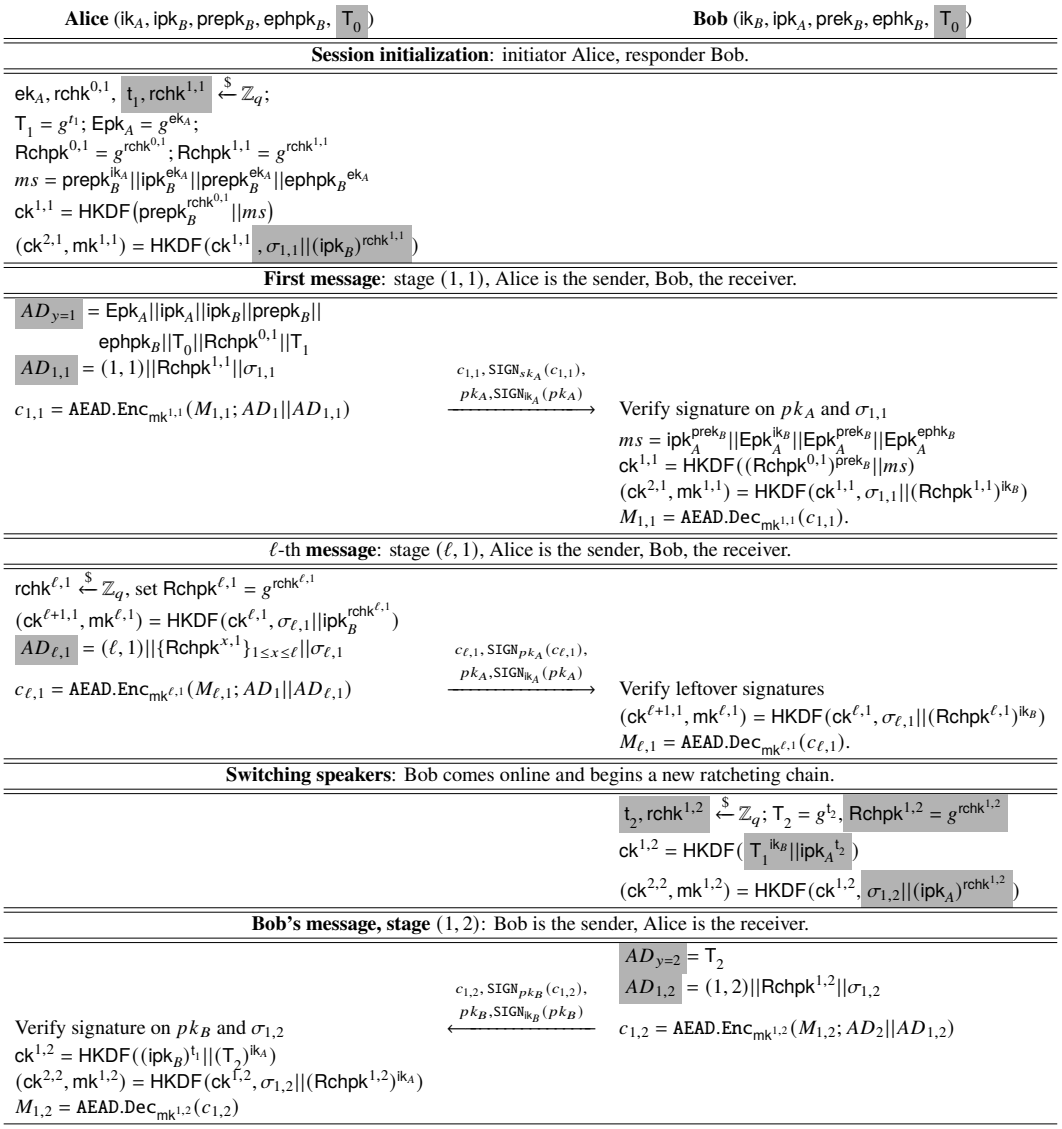


Figure 2.5 MARSHAL *protocol execution between Alice and Bob for the first few stages. The grey boxes indicate modifications with respect to Signal protocol [CGCD+17]. The transmitted data is also different but not in grey for more clarity.*

function along with ms to produce a chain key $\mathsf{ck}^{1,1}$.

Signature keys. A also needs a signature key (Section 2.3.1). We choose to use a second pair of signature keys, (sk_P, pk_P). If desired, these keys *could* coincide with $(\mathsf{ik}_P, \mathsf{ipk}_P)$– however, this is not compulsory. By differentiating those two pairs of keys, we allow future implementations to be somewhat agnostic of the underlying mathematical structure of the signature keys (whereas this is impossible for identity-keys, whose structure must support, *e.g.*, group exponentiations/scalar multiplications). Moreover, we limit the load on the centralized PKI server by not including the signature keys amongst the credentials stored for each party; instead users can authenticate those keys at session initialisation.

2.3.3 Communication phase

MARSHAL **ratcheting.** Our protocol heals faster than Signal because both parties ratchet asymmetrically at every stage. Thus, even at stage $(1, 1)$, Alice needs ratcheting randomness from Bob, which in MARSHAL comes in the form of the registered public key T_0 (see Section 2.3.1). For stages $(1, m)$ with integer $m \geq 1$, the party whose turn it is to speak will generate a cross-user ratcheting value t_m and compute the corresponding public value T_m. The T_m value is sent as part of the metadata of all messages with chain index m, and will be used for the ratchet at stage $(1, m + 1)$.

Moreover at each stage (ℓ, m) for $\ell, m \geq 1$, the current speaker also generates a same-user key-pair $(\mathsf{rchk}^{\ell,m}, \mathsf{Rchpk}^{\ell,m})$ which will be used to generate chain and message keys for stage $\mathsf{mk}^{\ell+1,m}$. To account for out-of-order messages the concatenation of all the public ratchet keys is included as metadata to each stage message.

MARSHAL **auxiliary data.** Each message is sent end-to-end encrypted, together with some additional metadata, which is meant to tell Bob how to run the key-schedule. At each stage (ℓ, m) with $\ell, m \geq 1$, the auxiliary value consists of two elements: $\mathsf{AD}_{y=m}$ and $\mathsf{AD}_{\ell,m}$. The former includes elements of the metadata that are universal across the chain (*i.e.*,all stages $(\cdot, m)$), whereas the second includes metadata that is stage-specific.

We detail each of the classes of stages (*cf.* Figure 2.5 and 2.6).

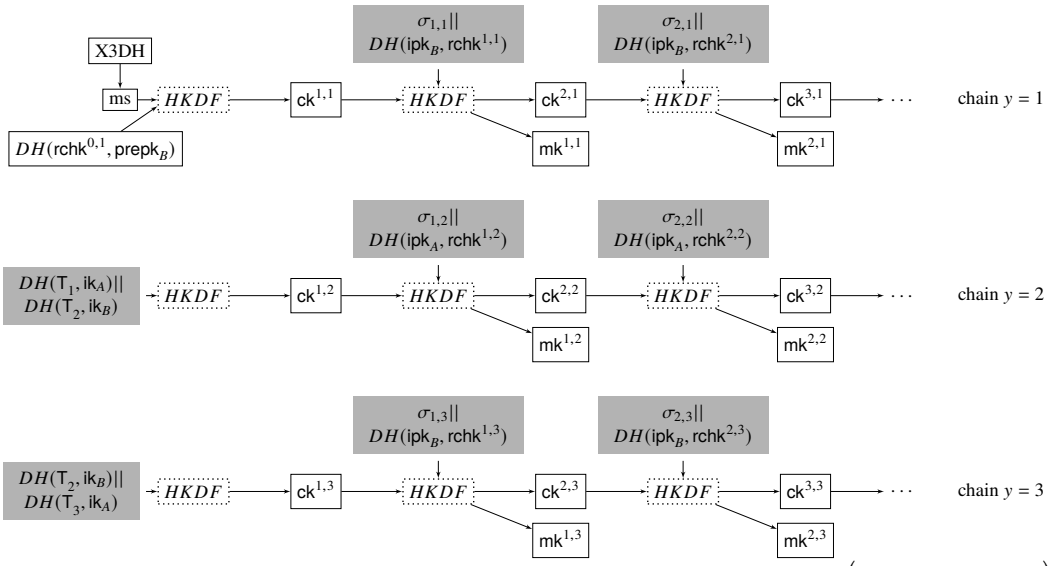


Figure 2.6 MARSHAL *key schedule diagram, where* $\sigma_{x,y} = \mathtt{SIGN}_{sk_A}\left(\mathsf{T}_{y-1}||\mathsf{Rchpk}^{x,y}\right)$ *for* y *odd and* $\sigma_{x,y} = \mathtt{SIGN}_{sk_B}\left(\mathsf{T}_{y-1}||\mathsf{Rchpk}^{x,y}\right)$ *for* y *even. The grey boxes indicate modifications with respect to Signal protocol [CGCD+17].*

Alice's first message. At session setup, Alice has generated its cross-user ratchet keys $(\mathsf{t}_1, \mathsf{T}_1)$, and computed the chain key $\mathsf{ck}^{1,1}$. Now she generates the same-user ratchet key $\mathsf{rchk}^{1,1}$ and computes $\mathsf{Rchpk}^{1,1} = g^{\mathsf{rchk}^{1,1}}$. The message and chain-keys are computed as follows $(\mathsf{ck}^{2,1}, \mathsf{mk}^{1,1}) \leftarrow \mathsf{HKDF}(\mathsf{ck}^{1,1}, \sigma_{1,1}||(\mathsf{ipk}_B)^{\mathsf{rchk}^{1,1}})$ where $\sigma_{1,1} := \mathtt{SIGN}_{sk_A}\left(T_0||\mathsf{Rchpk}^{1,1}\right)$. In the following, we will denote:

$$\sigma_{x,y} := \begin{cases} \mathtt{SIGN}_{sk_A}\left(\mathsf{T}_{y-1}||\mathsf{Rchpk}^{x,y}\right), \text{for } y \text{ odd} \\ \\ \mathtt{SIGN}_{sk_B}\left(\mathsf{T}_{y-1}||\mathsf{Rchpk}^{x,y}\right), \text{for } y \text{ even} \end{cases}$$

At chains $y = 1$ and $y = 2$, apart from cross-user ratchet keys, each user will need to include metadata that is universal for the session, and which helps at session setup. The metadata for $\mathsf{AD}_{y=1}$ includes public identity keys of Alice and Bob, medium-term and ephemeral keys of Bob as recovered by Alice from server, T_0 from the server, Alice's ephemeral public key used in the computation of the master secret, and two of Alice's ratchet public keys: its first same-user ratchet key $\mathsf{Rchpk}^{0,1}$, and its first cross-user ratchet key t_1. Finally, the stage-specific data contains: stage index $(1, 1)$ and same-user ratcheting public key $\mathsf{Rchpk}^{1,1}$. Alice computes $c_{1,1} = \mathtt{AEAD.Enc}_{\mathsf{mk}^{1,1}}(M_{1,1}; \mathsf{AD}_{y=1}||\mathsf{AD}_{1,1})$ and sends $c_{1,1}$, a signature on it, Alice's public signature key pk_A, and a signature on it.

Alice's $(\ell, 1)$ message, $\ell > 1$. Having already computed $\mathsf{t}_1, \mathsf{T}_1, \mathsf{AD}_{y=1}$, ratcheting material $\mathsf{Rchpk}^{1,1}, \mathsf{Rchpk}^{2,1}, \ldots, \mathsf{Rchpk}^{\ell-1,1}$, and the key $\mathsf{ck}^{l,1}$, Alice generates new same-user ratcheting key $\mathsf{rchk}^{\ell,1}$ and computes $\mathsf{Rchpk}^{\ell,1} = g^{\mathsf{rchk}^{\ell,1}}$. The key update relies on both long-term keys, for persistent authentication, and this same-user ratcheting key, for healing:

$$(\mathsf{ck}^{\ell+1,1}, \mathsf{mk}^{\ell,1}) \leftarrow \mathsf{HKDF}(\mathsf{ck}^{\ell,1}, \sigma_{\ell,1}||\mathsf{ipk}_B^{\mathsf{rchk}^{\ell,1}})$$

The stage-specific metadata consists of the stage $(\ell, 1)$ and *all the ratcheting keys* $\{\mathsf{Rchpk}^{x,1}\}_{1 \geq x \geq \ell}$. Then Alice computes $c_{\ell,1}$ and sends: the ciphertext, a signature on it, its signature public key, and a signature on that.

Note that this procedure applies to *all* messages (ℓ, m) for $\ell > 1$ and $m \geq 1$, in replacing the y stage-index above, from 1 to m.

Decryption (Bob side). When B comes online, he first needs to compute the same session-setup values as Alice, including the master secret ms and the first chain key $\mathsf{ck}^{1,1}$. To do so, B queries the server for A's registered identity key and verifies that it is identical to the one included in $\mathsf{AD}_{y=1}$. Then, B verifies the signature on pk_A, and, if the verification returns 1, it stores that key as A's signature key. From now on, B will use that key to verify A's signatures. In particular, the verification of pk_A is only done for the first message that Bob actually checks in the $y = 1$ chain. Once pk_A is validated, B retraces Alice's steps to compute ms, the chain keys, and eventually, the first message key. Then he uses authenticated decryption to decrypt the first message.

Bob's first message. B generates a new cross-user ratcheting value t_2 with corresponding public value T_2 and a same-user ratcheting key $\mathsf{rchk}^{1,2}$ and computes $\mathsf{Rchpk}^{1,2} := g^{\mathsf{rchk}^{1,2}}$.

Bob computes:

$$\mathsf{ck}^{1,2} \leftarrow \mathsf{HKDF}\left((T_1)^{\mathsf{ik}_B}||\mathsf{ipk}_A^{t_0}\right)$$

then its first sending keys:

$$(\mathsf{ck}^{2,2}, \mathsf{mk}^{1,2}) \leftarrow \mathsf{HKDF}(\mathsf{ck}^{1,2}, \sigma_{1,2}||(\mathsf{ipk}_A)^{\mathsf{rchk}^{1,2}})$$

Then analogously to Alice's first message, Bob splits the metadata into the two auxiliary values $\mathsf{AD}_{y=2}$ and $\mathsf{AD}_{1,2}$. The signed public key pk_B is also appended to each of the messages in stages with chain-index $y = 2$, *cf.* Figure 2.5.

Switching speakers. Similar computations will take place: generating cross-chain ratcheting public keys and new same-user ratcheting keys at every new message. The only differences with respect to stages $(1, 1)$ and $(1, 2)$ respectively will be that now the parties will no longer need to compute long-term keys or the master secret. In addition, starting from chain-index $y \geq 3$, the public key for signatures is no longer included in the message transmission.

Out-of-order messages/multiple messages. MARSHAL handles both out-of-order and lost messages to the same extent as Signal. Indeed, at each stage, the receiving party gets a list of ratcheting elements used along that chain, which will allow it to update correctly, even if some messages were lost in between. The parties will update their state in the order they receive the messages. In other words, say that Bob receives a message from Alice at stage $(1, 1)$, but then the next received message comes at stage $(4, 1)$ (thus, Bob is missing $(2, 1)$ and $(3, 1)$). Nevertheless, Bob will use the metadata at stage $(4, 1)$ to ratchet, thus computing the keys for stages $(2, 1)$ and $(3, 1)$ as well. If subsequently Bob receives message $(2, 1)$ with conflicting metadata, Bob disregards that.

In the same way, if multiple messages are received for some stage, the receiver will rely on the metadata (and message) received first, chronologically speaking. We give further details in 2.3.4.6.

2.3.4 Security Analysis

Our security models adapt and extends that of Blazy et al. [BBB+19]. We provide the general intuition for our framework and security games below, and then provide full proof.

The guarantees we want to prove for MARSHAL are AKE security (including authentication), post-compromise security, and out-of-order resilience within a fully adversarially controlled network. The first two properties make up a single security definition, written as a game between the *adversary* and the *challenger*. The adversary can register malicious users, corrupt users to obtain long-term secrets, reveal stage- and session-specific ephemeral values, access (a function of) the party's secret key as a black box, prompt new instances of existing parties, and send/receive messages. The adversary ultimately has to distinguish from random a real message key generated by an honest instance speaking with another honest instance.

The following theorem describes the security of MARSHAL in terms of PCS-AKE security and MLR-security. This security holds in the random oracle model (KDF are replaced by random oracles).

Theorem 1 *If the GDH [OP01] assumption holds, if the signature scheme employed is EUF-CMA-secure, then the* MARSHAL *protocol is PCS-AKE secure in the random oracle model (we model the two KDFs as $\mathcal{RO}_1, \mathcal{RO}_2$). In addition,* MARSHAL *is MLR-secure.*

We prove the security of our protocol with respect to a security model which is derived

from the identity-based setup of [BBB+19], rather than the one used by Cohn-Gordon *et al.* for their original analysis of Signal. This is chiefly because in [CGCD+17], Cohn-Gordon *et al.* bypass a feature of the protocol which we consider essential: sending metadata as AD attached to AEAD. Instead [CGCD+17] assumes that the metadata is sent unauthenticated. We prefer not to modify the protocol, and use the less composable security notion proposed (for the same reasons as we described here) by Blazy *et al.*. The proof of this theorem is not technically complex, but includes a lot of special cases (as in [CGCD+17]). This is a direct consequence of having excluded only trivial attacks from the winning conditions (given further). However, this was done in order to provide a more direct and honest comparison to the Signal protocol; indeed, with our winning conditions, Signal fails to attain security, whereas MARSHAL can be proved secure.

2.3.4.1 Syntax

We work in an environment with parties $P \in \mathcal{P}$, which have long-term key pairs $(\mathsf{sk}, \mathsf{pk})$, indexed by type. For instance in MARSHAL users have both identity and signature keys. Protocol *sessions* take part between two party *instances*. The i-th instance of P is denoted π_P^i. Beside long-term credentials, party instances also store:

pid: identifier of the instance's purported partner, denoted $\pi_P^i.\mathsf{pid}$.

sid: the session identifier $\pi_P^i.\mathsf{sid}$: an evolving set of instance-specific values[5].

sidsk: instance-specific private keys (like ephemeral session keys), denoted sidsk_P^i.

sidpk: the public keys sidpk_P^i corresponding to sidsk_P^i.

stages: a list with elements $(s, \cdot)$, associating stages $s = (x, y)$ to values $v \in \{0, 1\}$ depending on whether a message was received (1) or not (0). We write $s \in \pi_P^i$ if, and only if, $(s, v) \in \pi_P^i.\mathsf{stages}$.

Tr: the instance's transcript $\pi_P^i.T$, associating to each stage s all data sent or received at that stage (in plaintext) – denoted $\pi_P^i.T[s]$.

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.[6]. By default, each element of the list is set to the special value $\perp$. If the stage y is a sending stage for the party P, then this value remains $\perp$. We note this attribute as $\pi_P^i.\mathsf{rec}$.

var: a set $\pi_P^i.\mathsf{var}$ of ephemeral values used to compute stage keys, indexed by stage. For MARSHAL this includes rchk_P^s, mk^s and $\mathsf{ck}_{x-1,y}$.

While Signal only allows one conversation per pair of parties, MARSHAL permits multiple conversations between the same two peers. Compared to Blazy *et al.* [BBB+19], we add the private/public key-pair $(\mathsf{sidsk}, \mathsf{sidpk})$, which allows us to separate one-time session-specific randomness required for the master secret from stage-specific, recurring randomness. In addition, the same-user ratcheting at every stage compels us to extend $\pi.\mathsf{rec}$ to include *all the messages and metadata received*, not just the first one.

[5] In the case of our protocol, sid will be instantiated as a concatenation of all the auxiliary information sent throughout the protocol, ordered by stage.

[6] We adopt the corresponding notion from [BBB+19], in order to handle out-of-order messages. We note that in [BBB+19] this index is just a single value denoting *the first* received message. In our case, we need to extend this notion, since asymmetric ratcheting is done at every step.

As in Cohn-Gordon *et al.* [CGCD+17], we abstract the semi-trusted server from the setup, assuming it behaves honestly. It implies an authentication of each user upon registration. We define asynchronous messaging protocols `AsynchM` to be tuples of five algorithms (`aKeyGen`, `aStart`, `aRGen`, `aSend`, `aReceive`), such that:

$\underline{\texttt{aKeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})}$: outputs long-term credentials.

$\underline{\texttt{aStart}(P, \mathsf{role}, \mathsf{pid}) \to \pi_P^i}$: creates a new instance of (existing) party P with partner pid, such that P has a role $\mathsf{role} \in \{I, R\}$ (initiator/responder). The new instance is instantiated with that party's long-term keys.

$\underline{\texttt{aRGen}(1^\lambda) \to (\mathsf{rchk}, \mathsf{Rchpk})}$: outputs a public/private ratcheting key-pair.

$\underline{\texttt{aSend}(\pi_P^i, s, M, AD, aux) \to (\pi_P^i, C, AD^*, aux^*) \cup \perp}$: runs the sending part of the protocol for π_P^i on stage s, message M associated data AD, and auxiliary data aux; it outputs a ciphertext C with new associated and auxiliary data.

$\underline{\texttt{aReceive}(\pi_P^i, s, C, AD^*, aux^*) \to (\pi_P^i, M, AD, aux) \cup \perp}$: runs the receiving part of the protocol for instance π_P^i on stage s, ciphertext C, associated data AD^*, and auxiliary data aux^*; it outputs a message M and some (possibly transformed) associated and auxiliary data.

Definition 14 (Matching conversation) *Two instances π_A^i and π_B^j of an asynchronous-messaging protocol 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 $\pi_B^j.\mathsf{pid} = A$.*

Correctness. Assume π_A^i and π_B^j have matching conversation (A is the initiator). The protocol guarantees *correctness* if, in the absence of an adversary, for every stage $s = (x, y)$ with $s \in \pi_A^i$ and $s \in \pi_B^j$ it holds simultaneously:

- Both instances have identical keys $\mathsf{mk}^{x,y}$ and $\mathsf{ck}^{x-1,y}$ at stage s.
- Assuming $(\pi_P^u, s, C, AD^*, aux^*)$ was output by `aSend` on input (π_P^i, s, M, AD, aux) for $(P, u) \in \{(A, i), (B, j)\}$ if $(\pi_Q^v, s, C, AD^*, aux^*)$ is input to `aReceive`, then M is the message output by π_Q^v for $(Q, v) = \{(A, i), (B, j)\} \setminus (P, u)$.

2.3.4.2 Threat Model

We analyze MARSHAL with a fully adversarially-controlled network. The security properties we want to prove are confidentiality (in AKE model), authenticity (in AEAD model) and post-compromise (healing of the protocol). We capture these properties in a single security definition, by considering an attacker that can register malicious users, corrupt users to obtain their long-term secret key, reveal stage- and session-specific ephemeral values, access (a function of) the party's secret key as a black box, and manage communication by prompting parties to instantiate on new sessions, and by sending and receiving messages. We first give our unified security definition, then explain intuitively how it covers the properties mentioned above.

The adversary plays the security game against a *challenger*, which simulates all the honest parties. The attacker's goal is to distinguish from random a message key generated by an instance of an honest party, speaking to an instance of another honest party, for some

given stage. In order to formalize this game, instances will also need to keep track of the following attribute:

$\pi_P^Q.\mathsf{b}[s]$**:** the challenge bit takes a binary value chosen identically and independently at random at every new stage $s \in \pi_P^Q$ of an instance. The value of $\pi_P^Q.\mathsf{b}[s]$ for every stage s does not change, and is used by the test algorithm in our security game.

2.3.4.3 Oracles

Our security game will begin in the presence of an honest set of parties $\mathcal{P}$ of cardinality $\mathsf{n}_{\mathcal{P}}$, which are assumed to have long-term credentials $(\mathsf{sk}, \mathsf{pk})$ output by the key-generation algorithm `aKeyGen`. A set $\mathcal{P}^*$ is initialized to $\emptyset$ and will store maliciously-generated parties. We assume that each party $P \in \mathcal{P}$ has a unique identifier.

The adversary is given all the public keys, and will have access to the following oracles:

$\mathsf{oUReg}(P, \mathsf{pk}) \rightarrow \perp \cup \mathsf{OK}$: the user-registration oracle allows the adversary to register malicious parties. This oracles first checks that $P \notin \mathcal{P}$ – if the contrary is true, than the oracle returns a special symbol $\perp$. Else, the party P is added to $\mathcal{P}^*$ and the challenger stores its public key pk. The adversary receives a message OK.

$\mathsf{oReveal.XSid}(P, \mathsf{ktype}) \rightarrow \mathsf{sk} \cup \perp$: if $P \in \mathcal{P}$, then the corruption oracle reveals the long-term keys sk of the specified type ktype[7] of user P. Else the oracle returns a special symbol $\perp$.

$\mathsf{oStart}(P, \mathsf{role}, \mathsf{pid}) \rightarrow \pi_P^i \cup \perp$: on input a user identifier $P \in \mathcal{P}$, a $\mathsf{role} \in \{I, R\}$, and a partner identifier pid (indicating a user Q), this initialization oracle checks if P and pid are registered and if $P \in \mathcal{P}$, and if not, it returns $\perp$. For existing users, with $P \in \mathcal{P}$ it runs $\mathtt{aStart}(P, \mathsf{role}, \mathsf{pid})$ and returns the handle π_P^i to the adversary.

$\mathsf{oReveal}(\pi_P^i, \mathsf{ktypes}, s) \rightarrow \mathsf{key.set} \cup \perp$: on input a session identifier π_P^i, a list of non-longterm key types[8] included in a set ktypes, and a stage $s = (x, y)$, this revelation oracle first checks that $P \in \mathcal{P}$. If that is not the case, the oracle outputs $\perp$. Else, for each of the key types in var, the challenger first verifies that a value is stored for that key and stage[9]. If that is not so, the oracle outputs $\perp$ for the value of that key; else, it outputs the contents of that value.

$\mathsf{oAccessSK}(P, \mathsf{fct}, \mathsf{q.input}) \rightarrow \perp \cup \mathsf{q.rsp}$: on input a party P, some function fct, and a query $\mathsf{q.input}$ the black-box access oracle to (one of) the party's secret key(s)[10] first checks if $P \in \mathcal{P}$ (else, the oracle outputs $\perp$). If so, the oracle runs the function fct on sk and the additional input $\mathsf{q.input}$ and outputs $\mathsf{q.rsp}$.

$\mathsf{oTest}_b(\pi_P^i, s) \rightarrow \perp \cup K$: on input a party instance π_P^i and a stage s, this oracle checks if at least one of the following conditions is true:

- $P \in \mathcal{P}$; or

[7] In our protocol, for instance, users have two long-term keys, a public DH value and a signature key.

[8] Such key types could include ratcheting keys, chain keys, message keys, or any kind of session-specific or stage-specific key type.

[9] Note that we have two types of session-specific values stored for each instance: stage-specific values are stored by the attribute var, while session-specific ephemeral values are stored in sidsk_P^i.

[10] In our protocol each party will have two keys, one for signing and another long-term DH value. In our use of this oracle the function description fct will specify the function to compute (written as a text), and amongst the input to the query $\mathsf{q.input}$ we will actually have, as text, the name of the key that we require – either the signature, or the DH key.

- π_P^i does not exist; or
- $s \notin \pi_P^i$; or
- within the attribute var, the key $\mathsf{mk}^s = \perp$; or
- this oracle has already been called before, for either (π_P^i, s) or (π_Q^j, s) such that π_Q^j and π_P^i have matching conversation;

If at least one condition is true, then the oracle returns $\perp$. Otherwise, the oracle sets $b := \pi_P^i.\mathsf{b}[s]$. If $b = 1$, then the oracle retrieves the value mk^s from var and sets $K := \mathsf{mk}^s$. If $b = 0$, K is set to a random value from the key space.

$\mathsf{oSend}_b(\pi_P^i, s, AD) \rightarrow \perp \cup AD^*$: Like in [BBB+19], this oracle works in two modes: honest and adversary-driven, and its role is to prompt the given instance to send a message. If the input AD is set to $\perp$, then the oracle π_P^i will update honestly, whereas if $AD \neq \perp$, the oracle uses the information given by the adversary to update. More formally, the sending oracle first verifies that all the following conditions hold:

- $P \in \mathcal{P}$;
- π_P^i exists;
- P is the sender at stage s; *i.e.*, the y-component of s is odd if P was instantiated as an initiator for π_P^i, and y is even if P is the receiver in this session;
- $s \notin \pi_P^i$;
- π_P^i can actually update to stage $s = (x, y)$; that is, there exists a stage $s' \in \pi_P^i$ such that $s \in \mathsf{next}(s', \mathsf{sender})$;

If any of these conditions do not hold, the sending oracle returns $\perp$. Else, the oracle first checks if $AD = \perp$. If so, the oracle uses the honest ratchet-key generation algorithm to generate fresh ratcheting material, then calls the sending algorithm for the correct stage keys and a special message $\perp$. Finally in this case, we set a value $\mathsf{SHU}(\pi_P^i, s) := 1$, which indicates that this message was sent by an honest user. If, on the other hand, $AD \neq \perp$, the adversary interprets AD as a concatenation of M^*, sidsk^*, and var. It then simulates the sending algorithm by using values from the given data and generating random ones if some values are missing (as relevant for that stage[11]), and $\mathsf{SHU}(\pi_P^i, s)$ is set to 0. Finally, in both cases, we output a possibly modified associated data value AD^* output by the sending algorithm.

$\mathsf{oReceive}(\pi_P^i, s, AD) \rightarrow \perp$: once more this oracle works in two modes: honest and adversary-driven, and its goal is to allow the adversary to simulate what happens when a party instance receives a message. The oracle verifies that all the following conditions hold:

- $P \in \mathcal{P}$;
- π_P^i exists;
- P is the receiver at stage s;
- there exists $s' \in \pi_P^i$ such that $s \in \mathsf{next}(s', \mathsf{receiver})$;

If any of these conditions are untrue, then the oracle returns $\perp$. Then, the receiving algorithm is run on the input given to the oracle (including an empty ciphertext $\perp$ and the associated data value AD). If there exists an instance π_Q^j such that $Q \in \mathcal{P}$, and π_P^i

[11] For instance, in our protocol, some values stored in sidsk are only used for certain stages (notably the first stage (1,1)). If the stage input to the oracle differs from that stage, then the values in sidsk^* are ignored.

and π_Q^j have matching conversation, then if $\mathsf{SHU}(\pi_Q^j, s) = 1$ and C, AD were output by the oSend oracle for π_Q^j at stage s, then we set $\mathsf{RHU}(\pi_P^i, s) := 1$ (the message was honestly received at stage s). Else, we set $\mathsf{RHU}(\pi_P^i, s) := 0$.

Like in Blazy *et al.* [BBB+19], $\mathcal{A}$ is not given the true ciphertext, which can help $\mathcal{A}$ to distinguish the message keys. However, $\mathcal{A}$ can use oSend and its own, chosen input private keys to force parties to ratchet. Then $\mathcal{A}$ receives AD, a signature on AD (in aux), and other elements like sk_P and a signature on it.

2.3.4.4 Post-compromise security game

Our PCS AKE game is parametrized by the security parameter λ and a number $\mathsf{n}_{\mathcal{P}}$ of honest parties. The challenger starts by generating the $\mathsf{n}_{\mathcal{P}}$ honest parties and long-term secrets, giving the public keys and all system parameters to $\mathcal{A}$. The adversary can access all the oracles except oTest. At each new stage of each honest-party instance, the challenger generates a fresh test bit $\pi_P^i.\mathsf{b}[s^\star]$ for that instance and stage.

At some point, $\mathcal{A}$ outputs a party instance $\pi_P^\star$ and a stage $s^\star$. The challenger runs oTest on these inputs, outputting the returned key K (either the real message key of stage $s^\star$or a randomly-chosen key from the same key-space, depending on $\pi_P^i.\mathsf{b}[s^\star]$. Finally, $\mathcal{A}$ outputs a bit d, which is the adversary's guess for $b^\star$. We say $\mathcal{A}$ *wins* the experiment if $d = b^\star$ and if the adversary has played the game such that the active danger event $\mathsf{Act.Dan}(\mathsf{sid}, \mathsf{mk}^{x^\star, y^\star})$ as defined at below is not triggered. If the adversary terminates without outputting d, the challenger picks d uniformly at random, treating it as $\mathcal{A}$'s final output.

Definition 15 (PCS-AKE security) *Let* Π *be an asynchronous messaging protocol.* Π *is said to be* PCS-AKE secure *if for any polynomial time adversary* $\mathcal{A}$*, the adversary's advantage is negligibly close to 0 as a function of the security parameter. We define:*

$$\mathsf{Adv}_{\Pi}^{\mathsf{PCS-AKE}}(\mathcal{A}) := \left| \mathbb{P}[\mathcal{A} \textit{ wins } \mathsf{Exp}_{\Pi}^{\mathsf{PCS\text{-}AKE}}(\lambda, \mathcal{A})] - \frac{1}{2} \right|$$

where $\mathsf{Exp}_{\Pi}^{\mathsf{PCS\text{-}AKE}}(\lambda, \mathcal{A})$ *is defined in 2.7.*

Trivial attacks. We retroactively restrict $\mathcal{A}$'s otherwise full control of the oracles to rule out attacks that trivially break security. For instance, the oTest oracle is queried on a malicious party, it will win right away, since the protocol is designed to let both endpoints compute message keys. In general, such restrictions to the adversary indicate potential weaknesses in the protocol (which the adversary exploits): the more restrictions, the weaker the protocol's security.

In Signal querying $\mathsf{oReveal}$ for the chain and ratchet keys used at stage $s = (x, y)$ allows $\mathcal{A}$ to know: *all* chain and message keys for stages (x^*, y) with $x^* \geq x$; *and* all chain and message keys at stages $(x^*, y + 1)$ with $x \geq 1$. Moreover, $\mathcal{A}$ can hijack the conversation, using $\mathsf{oReceive}$ to learn future keys. These attacks are weaknesses specific to Signal, but they do not translate to MARSHAL. In fact, MARSHAL provides strictly stronger security than Signal.

As an example, consider the Signal protocol. Consider an adversary which casts a Reveal query for a party instance π_P^i at some stage $s = (x, y)$ for which P is the sender. The attacker

$\underline{\mathsf{Exp}_{\Pi}^{\text{PCS-AKE}}(\lambda, \mathcal{A})}$

$(\mathcal{P} = \{P_1, \cdots P_{\mathsf{n}_\mathcal{P}}\}) \leftarrow C(\lambda, \mathsf{n}_\mathcal{P})$
$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow C^{\mathtt{aKeyGen}(1^\lambda)} \quad \forall i \in \{1, \cdots, \mathsf{n}_\mathcal{P}\}$
$\mathcal{P}^* \leftarrow \emptyset$
$O_{\mathsf{PCS}} \leftarrow \left\{ \begin{array}{l} \mathsf{oUReg}(\cdot,\cdot), \mathsf{oReveal.XSid}(\cdot,\cdot), \mathsf{oStart}(\cdot,\cdot,\cdot), \mathsf{oReveal}(\cdot,\cdot,\cdot), \mathsf{oSend}(\cdot,\cdot,\cdot,\cdot), \\ \mathsf{oReceive}(\cdot,\cdot,\cdot,\cdot), \mathsf{oAccessSK}(\cdot,\cdot,\cdot), \mathcal{RO}_1(\cdot), \mathcal{RO}_2(\cdot) \end{array} \right\};$
$(\pi_P^\star, s^\star) \leftarrow \mathcal{A}^{O_{\mathsf{PCS}}}(1^\lambda)$
$K \leftarrow \mathsf{oTest}_{b\star}(\pi_P^\star, s^\star)$
$d \leftarrow \mathcal{A}^{O_{\mathsf{PCS}}}(\lambda, \mathsf{n}_\mathcal{P}, K)$

$\mathcal{A}$ wins iff. $d = b^\star$ and $\neg\mathsf{Act.Dan}(\pi_P^\star.\mathsf{sid}, \pi_P^\star.\mathsf{mk}^{s^*})$

Figure 2.7 *Description of the PCS AKE game, denoted* $\mathsf{Exp}_{\Pi}^{\text{PCS-AKE}}(\lambda, \mathcal{A})$ *between adversary* $\mathcal{A}$ *and challenger* C*. The game is parametrized by the security parameter* λ *and the number of honest parties* $\mathsf{n}_\mathcal{P}$*. The set of accessible oracles by* $\mathcal{A}$ *is denoted* O*.*

thus learns the chain key $\mathsf{ck}^{x,y}$ and the current private ratchet key $\mathsf{rchk}^{x,y}$. Due to the fact that in Signal, parties ratchet symmetrically along a single ratchet chain (while y remains constant), $\mathcal{A}$ is able to compute all chain and message keys for stages $s^* = (x^*, y^*)$ such that $y^* = y$ and $x^* \geq x$. In addition, the adversary can use its knowledge of $\mathsf{rchk}^{x,y}$ to also compute chain and message keys at stages $s^* = (x^*, y^*)$ such that $x \geq 1$ and $y^* = y + 1$. Finally, the adversary can also use its knowledge of these message keys in order to choose its own ratcheting material on chain $y^* = y + 2$ and successfully hijack the conversation. From this point, all future stages will be compromised. In the security analysis of Signal, security can only be proved if each of these attack avenues is closed to the adversary, *i.e.*, defined as a trivial attack. The adversary will, for instance, not be allowed to query oTest on a stage (x^*, y^*) if it has queried $\mathsf{oReveal}$ for the chain key $\mathsf{ck}^{x,y}$ such that $y^* = y$ and $x^* \geq x$, or if it has queried $\mathsf{oReveal}$ for the ratchet private key at stage (x, y) such that $y^* = y + 1$. However, as opposed to the case in the paragraph above, which applied by construction to *all* messaging protocols, the three attacks in the present paragraph are weaknesses of the Signal protocol only.

2.3.4.5 Winning conditions

Recall that the adversary's goal is to guess the test bit correctly for the stage and instance queried to oTest. However, we need to rule out trivial attacks, which we present as predicates (conjunctions and disjunctions of Boolean events). Each Boolean event is called a *danger* to a specific value, and we divide these into passive (compromising a party's state, but without hijacking) and active ($\mathcal{A}$ will also attempt hijacking) attacks.

Say $\mathcal{A}$ has queried oTest for instance π_A^i of initiator Alice, and stage $s^* = (x^*, y^*)$ (the winning conditions are mirrored for Bob). Assume π_A^i has session identifier sid, and let π_B^j be an honest instance with which π_A^i has matching conversation (party B is the partner of π_A^i).

Before giving the formal definition of our danger predicates, we illustrate in Figure 2.8 to Figure 2.14 the pictographic view of those danger to ease their understanding.

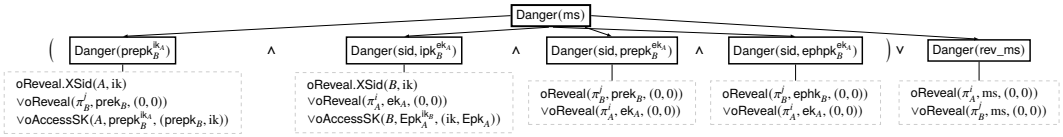


Figure 2.8 *Conditions for $\mathcal{A}$ to learn* ms.

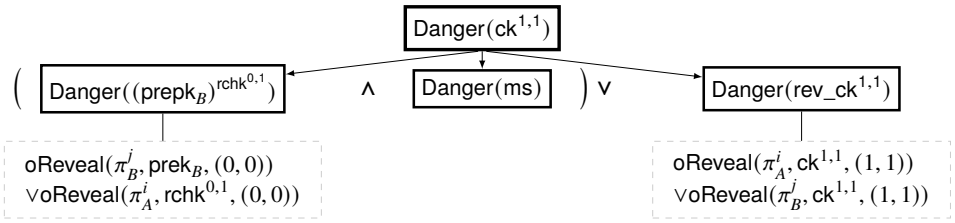


Figure 2.9 *Conditions for $\mathcal{A}$ to learn* ck^{1,1}.

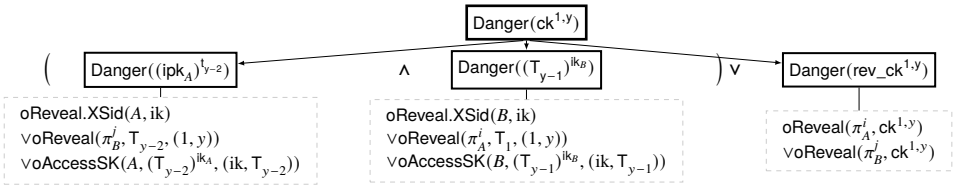


Figure 2.10 *Conditions for $\mathcal{A}$ to learn* ck^{1,y}.

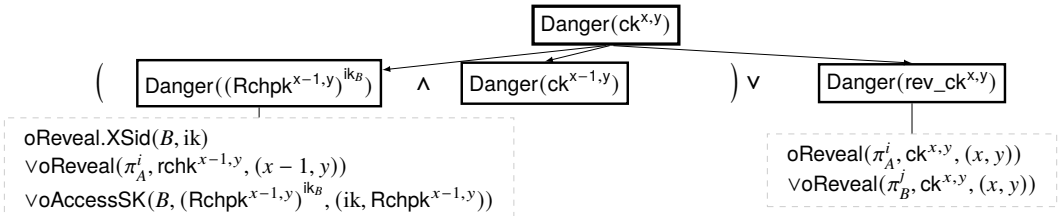


Figure 2.11 *Conditions for $\mathcal{A}$ to learn* ck^{x,y} *where y is odd (*i.e.*, Alice's chain).*

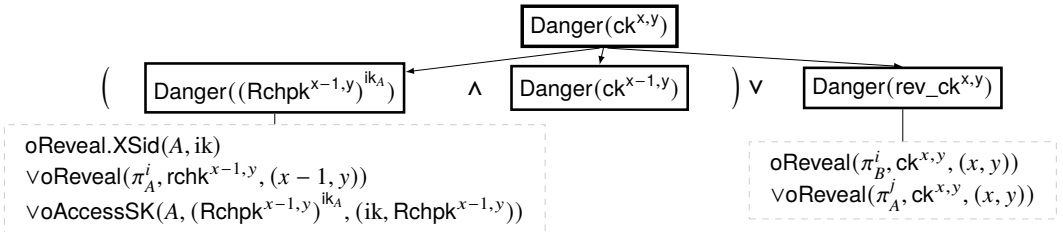


Figure 2.12 *Conditions for $\mathcal{A}$ to learn* ck^{x,y} *where y is even (*i.e.*, Bob's chain).*

We divide the adversary's trivial attacks into two categories. The first category includes ways in which the adversary can learn a key mk computed by an honest party in conversation with another honest party (passive attacker). The second category includes attacks in which the adversary attempts to hijack the conversation and thus dictate the keys of the conversation (active attacker). In that second case, the keys are shared by an honest entity and the

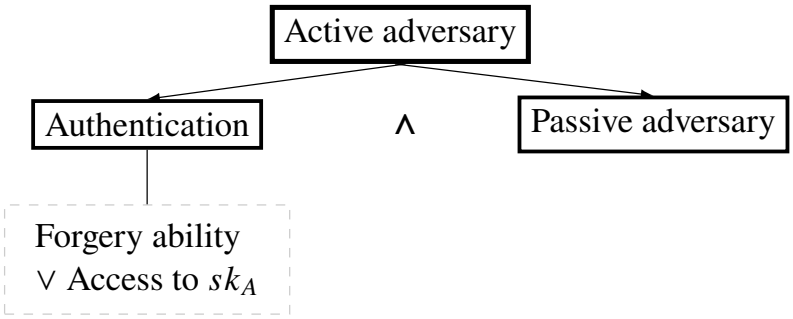


Figure 2.13 *General description of an active adversary.*

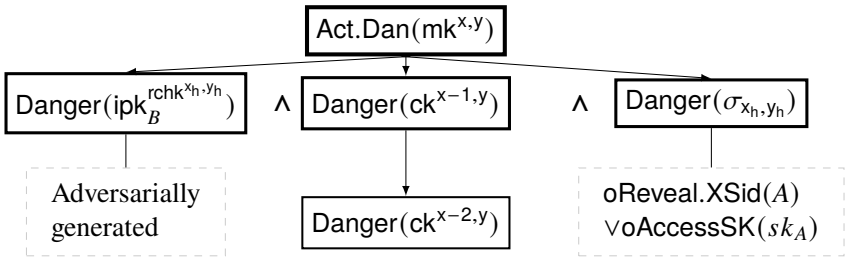


Figure 2.14 *Conditions for $\mathcal{A}$ to hijack the communication at stage $s = (x, y)$ with y_h odd.*

adversary.

Passive attacks. In the following, we assume that the adversary will ultimately input the values π_P^i and with $\pi_P^i.\mathsf{sid} = \mathsf{sid} \neq \perp$ and a value $s^* = (x^*, y^*)$ to oTest such that the oracle did not return $\perp$ (as per the conditions described above). Let us assume that $P = A$ and is the initiator of the protocol, whereas the partnered instance is B (the responder) [12].

- The adversary could learn mk^{s^*} trivially by just querying it to oReveal, either to π_A^i or to an instance π_B^j having matching conversation with π_A^i. The queries $\mathsf{oReveal}(\pi_A^i, \mathrm{mk}, s^*)$ and $\mathsf{oReveal}(\pi_B^j, \mathrm{mk}, s^*)$ are thus forbidden.
- The adversary can also learn the mk^{s^*} in session sid for a stage $s^* = (x^*, y^*)$ if it knows the input value to the HKDF computation that yields it. If y^* is odd, then the input values are of the form $\mathsf{ck}^{x^*,y^*} \| \sigma_{x^*,y^*} \| (\mathsf{Rchpk}^{x^*,y^*})^{\mathsf{ik}_B}$. If y^* is even, then the input to the HKDF is of the form $\mathsf{ck}^{x^*,y^*} \| \sigma_{x^*,y^*} \| (\mathsf{Rchpk}^{x^*,y^*})^{\mathsf{ik}_A}$. In each case, the adversary needs all the three values concatenated in the input. We focus below on each of these three values.

$\boxed{\mathsf{ck}^{x^*,y^*}}$: The adversary could learn this value by a number of different ways. The easiest way is for $\mathcal{A}$ to query oReveal for ck^{x^*,y^*} either from π_A^i (the challenge instance) or from a partnering instance π_B^j. Another way is for $\mathcal{A}$ to compute the value ck^{x^*,y^*} from sid itself.

1. Computing the value ms of the target session sid does not directly give ck^{x^*,y^*} for stage s^* of that session, but it may help. An adversary trivially learns this value if it knows $\mathsf{prepk}_B^{\mathsf{ik}_A} || \mathsf{ipk}_B^{\mathsf{ek}_A} || \mathsf{prepk}_B^{\mathsf{ek}_A} || \mathsf{ephpk}_B{}^{\mathsf{ek}_A}$. We define $\mathsf{Danger}(\mathsf{prepk}_B^{\mathsf{ik}_A})$ as the event that the adversary has issued (at least) one of the following queries: a oReveal.XSid query

[12] We make this assumption strictly in the interest of the legibility of the following analysis; the roles are in fact easily interchangeable.

for A, a oReveal query for prek_B, or an oAccessSK query for the exponentiation result. Similarly, $\mathsf{Danger}(\mathsf{sid}, \mathsf{ipk}_B^{\mathsf{ek}_A})$ is defined as the event that the adversary issues one of the following queries[13]: a oReveal.XSid query for B, a oReveal query for ek_A from session sid, or an oAccessSK query for the exponentiation. In addition, $\mathsf{Danger}(\mathsf{sid}, \mathsf{prepk}_B^{\mathsf{ek}_A})$ represents the event that $\mathcal{A}$ has issued one of the following queries: a oReveal query on the prek_B value used in sid or a oReveal query on ek_A from sid. Furthermore, $\mathsf{Danger}(\mathsf{sid}, \mathsf{ephpk}_B{}^{\mathsf{ek}_A})$ is the event that the adversary issues one of the following queries: a oReveal query for ephk_B from sid or a oReveal query for ek_A from sid. Finally, $\mathsf{Danger}(\mathsf{rev\_ms})$ is the event that the adversary issues one of the following queries: either $\mathsf{oReveal}(\pi_A^i, \mathrm{ms})$ or $\mathsf{oReveal}(\pi_B^i, \mathrm{ms})$ We define $\mathsf{Danger}(\mathsf{sid}, \mathsf{ms})$ as the event that all the following events occur: $\mathsf{Danger}(\mathsf{prepk}_B^{\mathsf{ik}_A})$, $\mathsf{Danger}(\mathsf{sid}, \mathsf{ipk}_B^{\mathsf{ek}_A})$, $\mathsf{Danger}(\mathsf{sid}, \mathsf{prepk}_B^{\mathsf{ek}_A})$, $\mathsf{Danger}(\mathsf{sid}, \mathsf{ephpk}_B{}^{\mathsf{ek}_A})$, or $\mathsf{Danger}(\mathsf{rev\_ms})$.

This is formally defined in Definition 16.

2. If $x^* = 1$ and $y^* = 1$, then an adversary can trivially compute the $\mathsf{ck}^{1,1}$ value of sid if it knows ms and $(\mathsf{prepk}_B)^{\mathsf{rchk}^{0,1}}$. We define $\mathsf{Danger}(\mathsf{sid}, (\mathsf{prepk}_B)^{\mathsf{rchk}^{0,1}})$ to be the event that the adversary issues one of the following queries: a oReveal query for prek_B used in sid or a oReveal query for $\mathsf{rchk}^{0,1}$ in sid. Then, let $\mathsf{Danger}(\mathsf{sid}, \mathsf{ck}^{1,1})$ be the event that both these events occur for the target session sid: $\mathsf{Danger}(\mathsf{sid}, \mathsf{ms})$ (see the previous bullet point) and $\mathsf{Danger}(\mathsf{sid}, (\mathsf{prepk}_B)^{\mathsf{rchk}^{0,1}})$.

3. If $x^* = 1$ and $y^* > 1$, then an adversary can trivially compute ck^{x^*,y^*} if it knows $(\mathsf{T}_1)^{\mathsf{ik}_B} || (\mathsf{ipk}_A)^{\mathsf{t}_0}$. We define $\mathsf{Danger}(\mathsf{sid}, (\mathsf{T}_1)^{\mathsf{ik}_B})$ to be the event that the adversary makes at least one of these three queries: a oReveal.XSid query for ik_B, a oReveal query for the t_1 value used in sid, or an oAccessSK query to find the result of the exponentiation. Similarly, $\mathsf{Danger}(\mathsf{sid}, (\mathsf{ipk}_A)^{\mathsf{t}_0})$ is defined as the event that the adversary issues at least one of the following queries: a oReveal.XSid query for the key ik_A of party A, a oReveal query for t_0, or an oAccessSK query for the exponentiation. Then, let $\mathsf{Danger}(\mathsf{sid}, \mathsf{ck}^{\mathsf{x}^*,\mathsf{y}^*})$ be defined as the event that both these events occur: $\mathsf{Danger}(\mathsf{sid}, \mathsf{T}_1)^{\mathsf{ik}_B})$ and $\mathsf{Danger}(\mathsf{sid}, (\mathsf{ipk}_A)^{\mathsf{t}_0})$.

4. If $x^* > 1$ and y^* is odd, then the adversary can trivially compute ck^{x^*,y^*} if it knows $\mathsf{ck}^{x^*-1,y^*} || \sigma_{x^*-1,y^*} || (\mathsf{Rchpk}^{x^*-1,y^*})^{\mathsf{ik}_B}$ (recall that A is the initiator and B is the responder). The value σ_{x^*-1,y^*} is just a signed value on publicly-known keys (present in the AD of the ciphertext), thus the adversary already knows this input. Additionally, we define $\mathsf{Danger}(\mathsf{sid}, (\mathsf{Rchpk}^{\mathsf{x}^*-1,\mathsf{y}^*})^{\mathsf{ik}_B})$ to be the event that the adversary issues at least one of the following queries: a oReveal.XSid query for ik_B, a oReveal query for $\mathsf{rchk}^{x^*-1,y^*}$ from either π_A^i or π_B^j, or an oAccessSK query for the exponentiation result. Finally, we must look at the trivial attacks allowing the adversary to learn ck^{x^*-1,y^*}. This last part must be defined recursively: notably $\mathsf{Danger}(\mathsf{sid}, \mathsf{ck}^{\mathsf{x}^*-1,\mathsf{y}^*})$ is the event that all the

[13]Note that when we defined $\mathsf{Danger}(\mathsf{prepk}_B^{\mathsf{ik}_A})$, the session identifier sid was not given as a parameter to that value; yet, when we defined $\mathsf{Danger}(\mathsf{sid}, \mathsf{ipk}_B^{\mathsf{ek}_A})$ with sid as a parameter. This is because the former endangered value ($\mathsf{prepk}_B^{\mathsf{ik}_A}$) contains long- and medium-term values (which can be obtained from *any* session that involves A and B within some time-space, for which prepk_B is valid); by contrast, the latter value includes the session-specific ek_A and we thus need the value sid to pinpoint which value ek_A we mean.

following events occur: $\mathsf{Danger}(\mathsf{sid}, (\mathsf{Rchpk}^{x^*-1,y^*})^{\mathsf{ik}_B})$, and either an oReveal query for ck^{x^*-1,y^*} or – if $x^* > 2$ – $\mathsf{Danger}(\mathsf{sid}, \mathsf{ck}^{x^*-2,y^*})$.

5. If $x^* > 1$ and y^* is even, then the adversary can trivially compute ck^{x^*,y^*} if it knows $\mathsf{ck}^{x^*-1,y^*} \| \sigma_{x^*-1,y^*} \| (\mathsf{Rchpk}^{x^*-1,y^*})^{\mathsf{ik}_A}$. Again, the signature is known. Furthermore, $\mathsf{Danger}(\mathsf{sid}, (\mathsf{Rchpk}^{x^*-1,y^*})^{\mathsf{ik}_A})$ denotes the event that the adversary issues at least one of the following queries: a oReveal.XSid query for ik_A, a oReveal query for $\mathsf{rchk}^{x^*-1,y^*}$ from either π_A^i or π_B^j, or an oAccessSK query for the exponentiation result. Finally, we must look at the trivial attacks allowing the adversary to learn ck^{x^*-1,y^*}. This last part must be defined recursively: notably $\mathsf{Danger}(\mathsf{sid}, \mathsf{ck}^{x^*-1,y^*})$ is the event that all the following events occur: $\mathsf{Danger}(\mathsf{sid}, (\mathsf{Rchpk}^{x^*-1,y^*})^{\mathsf{ik}_A})$, and either an oReveal query for ck^{x^*-1,y^*} or – if $x^* > 2$ – $\mathsf{Danger}(\mathsf{sid}, \mathsf{ck}^{x^*-2,y^*})$.

$\boxed{(\mathsf{Rchpk}^{x^*,y^*})^{\mathsf{ik}_B}}$: For an odd value of y^*, we define $\mathsf{Danger}(\mathsf{sid}, (\mathsf{Rchpk}^{x^*,y^*})^{\mathsf{ik}_B})$ to be the event that the adversary makes at least one of the following queries: a oReveal.XSid query on ik_B, a oReveal query on rchk^{x^*,y^*}, or an oAccessSK query for the result of the exponentiation. If y^* is even, this value is undefined.

$\boxed{(\mathsf{Rchpk}^{x^*,y^*})^{\mathsf{ik}_A}}$: For an even value of y^*, we define $\mathsf{Danger}((\mathsf{Rchpk}^{x^*,y^*})^{\mathsf{ik}_A})$ as the event that the adversary makes at least one of the following queries: a oReveal.XSid query for A, a oReveal query on rchk^{x^*,y^*}, or an oAccessSK for the result of the exponentiation. If y^* is odd, this value is undefined.

Definition 16 ($\mathsf{Danger}(\mathsf{sid}, \mathsf{ms})$) *We define the event that the master secret of some target session* sid *is known to the adversary by* $\mathsf{Danger}(\mathsf{sid}, \mathsf{ms})$*, notably as follows:*

$$\begin{aligned}\mathsf{Danger}(\mathsf{sid}, \mathsf{ms}) = \Big(&\mathsf{oReveal}(\pi_A^i, ms) \vee \mathsf{oReveal}(\pi_B^i, ms)\Big)\\ &\wedge \mathsf{Danger}(\mathsf{prepk}_B^{\mathsf{ik}_A})\\ &\wedge \mathsf{Danger}(\mathsf{sid}, \mathsf{ipk}_B^{\mathsf{ek}_A})\\ &\wedge \mathsf{Danger}(\mathsf{sid}, \mathsf{prepk}_B^{\mathsf{ek}_A})\\ &\wedge \mathsf{Danger}(\mathsf{sid}, \mathsf{ephpk}_B{}^{\mathsf{ek}_A}),\end{aligned}$$

with :

$$\begin{aligned}\mathsf{Danger}(\mathsf{prepk}_B^{\mathsf{ik}_A}) = \ &\mathsf{oReveal.XSid}(A, ik)\\ &\vee \mathsf{oReveal}(\pi_B^j, \mathsf{prek}_B, (0,0))\\ &\vee \mathsf{oAccessSK}(A, \mathsf{prepk}_B^{\mathsf{ik}_A}, (\mathsf{prepk}_B, ik))\end{aligned}$$

$$\begin{aligned}\mathsf{Danger}(\mathsf{sid}, \mathsf{ipk}_B^{\mathsf{ek}_A}) = \ &\mathsf{oReveal.XSid}(B, ik)\\ &\vee \mathsf{oReveal}(\pi_A^i, \mathsf{ek}_A, (0,0))\\ &\vee \mathsf{oAccessSK}(B, \mathsf{Epk}_A^{\mathsf{ik}_B}, (ik, \mathsf{Epk}_A))\end{aligned}$$

$$\begin{aligned}\mathsf{Danger}(\mathsf{sid}, \mathsf{prepk}_B^{\mathsf{ek}_A}) = \ &\mathsf{oReveal}(\pi_B^j, \mathsf{prek}_B, (0,0))\\ &\vee \mathsf{oReveal}(\pi_A^i, \mathsf{ek}_A, (0,0))\end{aligned}$$

$$\mathsf{Danger}(\mathsf{sid}, \mathsf{ephpk}_B{}^{\mathsf{ek}_A}) = \mathsf{oReveal}(\pi_B^j, \mathsf{ephk}_B, (0,0)) \\ \vee \mathsf{oReveal}(\pi_A^i, \mathsf{ek}_A, (0,0)).$$

*We note that the reveal queries take in input the value of the stage at which the ephemeral values were generated (*i.e.*, stage (0,0)).*

Summarizing the analysis above, we define by $\mathsf{Danger}(\mathsf{sid}, \mathsf{mk}^{x^*,y^*})$ as follows: if y^* is odd, then $\mathsf{Danger}(\mathsf{sid}, \mathsf{mk}^{x^*,y^*})$ is the event that the adversary issues a oReveal query for mk^{x^*,y^*} or alternatively that it triggers *all* the following events: $\mathsf{Danger}(\mathsf{sid}, (\mathsf{Rchpk}^{x^*,y^*})^{\mathsf{ik}_B})$ and $\mathsf{Danger}(\mathsf{sid}, \mathsf{ck}^{x^*,y^*})$; if y^* is even, then $\mathsf{Danger}(\mathsf{sid}, \mathsf{mk}^{x^*,y^*})$ is the event that the adversary issues a oReveal query for mk^{x^*,y^*} or alternatively that it triggers *all* the following events: $\mathsf{Danger}(\mathsf{sid}, (\mathsf{Rchpk}^{x^*,y^*})^{\mathsf{ik}_A})$ and $\mathsf{Danger}(\mathsf{sid}, \mathsf{ck}^{x^*,y^*})$.

Active attacks. We now concentrate on active attacks, in which the adversary attempts to know the keys by being a legitimate end point at the target stage. Interestingly, while targeting an instance of a maliciously-registered party (or an instance partnering such an instance) immediately gives the adversary all the keys in that session, corruption does not have the same effect. In the following, we focus on active attacks by the adversary, which will invariably involve both long-term and ephemeral keys of previous stages.

Let $s^* = (x^*, y^*)$ be the target stage input by the adversary to its oTest query, and let π_P^i be the party instance targeted by this query. We will require several additional restrictions.

- As defined at the beginning of this section, the adversary may not query oTest on an instance of a malicious party, nor on an instance partnering a malicious party (*i.e.*, we declare the adversary to lose if it has triggered $\mathsf{Danger}(\pi_P^i)$).
- The first bullet point ensures the adversary cannot be an endpoint of the targeted conversation. However, $\mathcal{A}$ could in fact only *hijack* a session and thus learn some values. In Signal, successful hijacking leads to the adversary controlling the full conversation. For MARSHAL, healing occurs quickly except for a few extreme cases, which we rule out below.

We describe how $\mathcal{A}$ can hijack the communication between A and B, *i.e.*, how $\mathcal{A}$ can use its own keying material to impersonate one of the communicating partners.

We say a stage $s_h = (x_h, y_h)$ is *hijacked* if for that stage the adversary actively impersonates the sending party, *i.e.*,it sends (via oReceive) the receiving party a message that: (i) was not actually sent by the sender; and simultaneously (ii) includes ephemeral information that diverges from the state of the actual sender. In other words, if the adversary takes a message sent by the actual sender, then queries the signature oracle for a fresh signature on the same data (thus the messages now differ because the signature would not be the same), this does not count as hijacking (condition (i) is satisfied, but not condition (ii)). However, if the adversary inserts new ephemeral values in the AD of the message for that stage, then somehow obtains a correct signature, then this counts as hijacking. Formally, we have the following definition.

Definition 17 *Let $s_h := (x_h, y_h)$ be a communication stage of a protocol session* sid*, for*

which we assume without loss of generality that party A is the sender (through instance π_A^i) and B is the receiver (through instance π_B^j). We say that $\mathcal{A}$ has hijacked *stage s_h if, and only if, the following conditions hold simultaneously:*

- *$\mathcal{A}$ has queried* $\mathsf{oReceive}(\pi_B^j, s_h, AD_h)$*;*
- *the value AD_h was never output by an* $\mathsf{oSend}(\pi_A^i, s_h, \cdot)$ *query;*
- *there exists a value $v \in AD_h$, but such that $v \notin \mathsf{sidsk}_A^i \cup \pi_i^i.\mathsf{var}[s_h]$.*

We call stage s_h successfully *hijacked if, in addition to the conditions above, it also holds that the* oReceive *query in the first bullet point has yielded an output that is different from $\perp$.*

For active attacks, our adversary will have to first hijack an initial stage. In Signal, if the adversary succeeds in doing this for one specific stage, it will immediately – without access to long-term credentials – gain access to a number of future stages. By comparison, in MARSHAL the adversary only gains trivial access to stage keys if it directly attacks each stage independently or obtains long-term access to the user's long-term keys.

Assume that the tested stage is $s^* := (x^*, y^*)$ and that the adversary has hijacked the session at some stage $s_h := (x_h, y_h)$ (without loss of generality, by compromising party party A, the sender at stage s_h). We define the active danger to the message key of stage s^* (denoted $\mathsf{Act.Dan}(\mathsf{sid}, \mathsf{mk}^{x^*,y^*})$) as a *complementary event* to the danger event previously defined for a passive adversary. In other words, both types of sequences of queries will be ruled out when considering the adversary's success.

Let $\mathsf{Hijack}(\mathsf{sid}, \mathsf{s_h})$ be the event that the adversary *successfully* hijacked session sid. We note that there are many different values the adversary could replace, each leading to a complex chain of events that would need to be triggered to actively endanger mk^{x^*,y^*}. In the interest of legibility, we reduce these chains of events to only events that legitimately allow the adversary direct access to that message key: reveal queries on it, access to signatures, or access to identity keys. In particular, we are ruling out more attacks than strictly necessary, thus weakening the adversary. However, note that in so doing, we stress the one aspect of our protocol that distinguishes it from Signal. In our protocol, the adversary will need to keep accessing long-term keys from Alice or Bob in order to endanger the message key (or query reveal for that key directly).

Let party A be the sender at stage s_h of a session sid such that $\pi_A^i.\mathsf{sid} = \mathsf{sid}$, and let π_B^j be partnering π_A^i (if such a partner exists). We divide our analysis of $\mathsf{Act.Dan}(\mathsf{sid}, \mathsf{mk}^{x^*,y^*})$ (given oTest is queried on stage s^*) into two cases (for clarity):

> $y^* = y_h + 2s$, where $s \in \mathbb{N}$ and s^* comes after s_h[14]. In other words, $\mathcal{A}$ tests a stage for which party A (the party $\mathcal{A}$ impersonated during hijacking) is the sender. We define $\mathsf{Act.Dan}(\mathsf{sid}, \mathsf{mk}^{x^*,y^*})$ to be the event that the following events occur: $\mathsf{Hijack}(\mathsf{sid}, \mathsf{s_h})$, the adversary has queried oTest for π_B^j, *and* (at least) one of the two following: a oReveal.XSid query for sk_A, *or* oAccessSK access to $h(c^*)$ (receiving σ) such that $\mathcal{A}$ has queried oReceive with input $\pi_B^j, s^*, (\mathsf{AD}, \sigma)$ with $\mathsf{AD} \in c^*$.
>
> $y^* = y_h + 2s + 1$, where $s \in \mathbb{N}$ and s^* comes after s_h. In other words, $\mathcal{A}$ tests a stage for which party B (the party to which $\mathcal{A}$ impersonated A during the hijack) is the sender. We define $\mathsf{Act.Dan}(\mathsf{sid}, \mathsf{mk}^{x^*,y^*})$ to be the event that the following events

[14] Concretely, $y^* \geq y_h$ and if $y^* = y_h$, then $x^* \geq x_h$.

occur: $\mathsf{Hijack}(\mathsf{sid}, \mathsf{s_h})$, the adversary has queried oTest for π_B^j, *and* (at least) one of the two following: a oReveal.XSid query for ik_A, *or* oAccessSK access to $(R^*)_A^{\mathsf{ik}}$ such that $\mathcal{A}$ has received (AD, σ) as output of an oSend query with input $\pi_B^j, s^*, \perp$.

2.3.4.6 Message-loss resilience

For the message-loss resilience game we need an additional notion. Let π_P^i be such that $s \in \pi_P^i$ for some stage s. We denote by $\mathsf{next}(\pi_P^i, s)$ the stages reachable for that instance from stage s. This depends on P's role (sender or receiver) at stage $s = (x, y)$. If P was a sender, then $\mathsf{next}(\pi_P^i, s) = \{(x', y) | x' \geq x + 1\}$. If P was a receiver, then $\mathsf{next}(\pi_P^i, s) = \{(x'', y + 1) | x'' \geq 1\}$.

The message-loss resilience game begins like PCS-AKE, by creating $\mathsf{n}_\mathcal{P}$ honest parties and their long-term credentials. We provide an illustration of it in Figure 2.15. The adversary receives the public keys, then gets access to the oracles oStart, oReveal.XSid, oReveal, oAccessSK, and a crippled version of oSend and oReceive ($\mathcal{A}$ must always use these oracles in honest mode). The adversary finally stops, outputting a protocol instance/stage pair π_P^i, s^*, for which the challenger must produce the key mk^{s^*} (or $\perp$ if it does not know it). We say $\mathcal{A}$ *wins* if, and only if, the challenger has output $\perp$ and there exists a stage $s \in \pi_P^i$ such that $s^* \in \mathsf{next}(\pi_P^i, s)$.

$\underline{\mathsf{Exp}_\Pi^{\mathsf{MLR}}(\lambda, \mathcal{A})}$

$(\mathcal{P} = \{P_1, \cdots P_{\mathsf{n}_\mathcal{P}}\}) \leftarrow C(\lambda, \mathsf{n}_\mathcal{P})$
$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow C^{\mathtt{aKeyGen}(1^\lambda)} \quad \forall i \in \{1, \cdots, \mathsf{n}_\mathcal{P}\}$
$O_{\mathsf{MLR}} \leftarrow \left\{ \begin{array}{l} \mathsf{oReveal.XSid}(\cdot,\cdot), \mathsf{oStart}(\cdot,\cdot,\cdot), \mathsf{oReveal}(\cdot,\cdot,\cdot), \mathsf{oSend}^*(\cdot,\cdot,\cdot,\cdot), \\ \mathsf{oReceive}^*(\cdot,\cdot,\cdot,\cdot), \mathsf{oAccessSK}(\cdot,\cdot,\cdot) \end{array} \right\};$
$(\pi_P^\star, s^\star) \leftarrow \mathcal{A}^{O_{\mathsf{MLR}}}(1^\lambda)$
$\mathsf{mk} \leftarrow C$

$\mathcal{A}$ wins iff. $\mathsf{mk} \neq \perp$

Figure 2.15 *Description of the MLR game, denoted* $\mathsf{Exp}_\Pi^{\mathsf{MLR}}(\lambda, \mathcal{A})$ *between adversary* $\mathcal{A}$ *and challenger* C*. The game is parametrized by the security parameter* λ *and the number of honest parties* $\mathsf{n}_\mathcal{P}$*. Oracles* oSend* *and* oReceive* *are the crippled versions of the original* oSend *and* oReceive*, restricted to honest mode only.*

2.3.4.7 Security proofs

We give the proof of theorem 1:

Theorem 1 *If the GDH [OP01] assumption holds, if the signature scheme employed is EUF-CMA-secure, then the* MARSHAL *protocol is PCS-AKE secure in the random oracle model (we model the two KDFs as* $\mathcal{RO}_1, \mathcal{RO}_2$*). In addition,* MARSHAL *is MLR-secure.*

Proof 1 *The security statement is parametrized by the maximal number of stages* $\mathsf{n_S}$ *run by any given instance, the number of parties generated by the adversary* $\mathsf{n_P}$*, the number*

of medium-term keys $\mathsf{n}_{\mathsf{prek}}$ *deployed, and the number of instances* n_π *created by any given party.*

We prove the statement in a sequence of game hops, as follows.

$\mathbb{G}_0$: *This is the* $\mathsf{Exp}^{\mathsf{PCS\text{-}AKE}}_{\Pi}(\lambda, \mathcal{A})$ *described in Figure 2.7. We denote by* Adv_0 *the maximal advantage an adversary* $\mathcal{A}$ *to win this game.*

$\mathbb{G}_1$: *This game is identical to* $\mathbb{G}_0$*, except, whenever prompted to generate randomness within a protocol step, any honestly-created instance of the protocol will produce unique, random values. In other words, we remove the probability of having a collision for any of the private material, including:*

- *Long-term keys:* ik*,* sk*;*
- *Medium-term keys:* prek*,* t_0*;*
- *Session-setup ephemeral keys:* ek*,* $\mathsf{rchk}^{0,1}$*;*
- *Stage-specific ephemeral keys: same-user ratchet keys* $\mathsf{rchk}^{x,y}$ *with* $x \geq 0$*, and, at every fresh* $y > 0$*, a new cross-user ratcheting key* t_y*.*

For ease of notation, we will upper-bound the number of fresh cross-user ratcheting keys by the total number of stages n_S *(the bound is tight if we have only one message per chain). Let* Adv_1 *be the advantage of the adversary* $\mathcal{A}$ *in this game.*

It holds that:

$$\mathsf{Adv}_0 \leq \frac{\binom{\mathsf{n_P}+\mathsf{n_P}\cdot\mathsf{n}_{\mathsf{prek}}+\mathsf{n}_\pi\cdot(2+2\mathsf{n_S})}{2}}{q} + \mathsf{Adv}_1 \ .$$

$\mathbb{G}_2$: *This game is identical to* $\mathbb{G}_1$*, except that the challenger guesses and outputs (privately w.r.t. the adversary) a tuple consisting of an instance* π^i_P *and a stage index* s^**. The game is lost if these do not coincide with the tuple output by the adversary to* oTest*. Let the advantage of the adversary in this hop be* Adv_2*. Then:*

$$\mathsf{Adv}_1 \leq \frac{1}{\mathsf{n_P}\mathsf{n}_\pi\mathsf{n_S}}\mathsf{Adv}_2.$$

Note that the challenger's guess π^i_P, s^* *gives it more information than just the target instance and stage. In particular, the challenger now knows: the target party P, the role of P in the target session (initiator/responder), the role of party P in* π^i_P *at stage* s^* *(sender/receiver), as well as the role of P at every prior or future stage in that conversation.*

$\mathbb{G}_3$: *This game hop is identical to* $\mathbb{G}_2$ *except that the challenger will now refuse to answer the adversary's hijacking queries (*$\mathsf{oReceive}$ *queries with adversarially-chosen input substituted for output of honest* oSend *queries) for the following queries:*

- *If* $\mathcal{A}$ *queries* $\mathsf{oReceive}$ *maliciously for an instance of some party other than P;*
- *If* $\mathcal{A}$ *queries* $\mathsf{oReceive}$ *maliciously for an instance of P other than* π^i_P*;*
- *If* $\mathcal{A}$ *queries* $\mathsf{oReceive}$ *maliciously for the target instance* π^i_P *for a stage s that comes after* s^**;*
- *If* $\mathcal{A}$ *queries* $\mathsf{oReceive}$ *maliciously for* π^i_P*, such that: s precedes* s^* *and* π^i_P *already has a message key* mk *set for the target stage or any other stage in that chain at the time of the* $\mathsf{oReceive}$ *query.*

We argue that the adversary's advantage in this game, denoted Adv_3*, is undiminished with respect to* $\mathbb{G}_2$ *, because: (1) as of* $\mathbb{G}_1$ *we have unique session identifiers; (2) we are using random oracles for the key derivation; (3) once key material is accepted*

for the target stage (or an ulterior one in the same stage), that automatically sets the ratcheting information, which cannot be reset by the malicious oReceive *query. Thus:*

$$\mathsf{Adv}_2 = \mathsf{Adv}_3;$$

Note that, starting from this game, the only hijacking attempts that will work are those for the target instance, for stages prior to the target stage.

$\mathbb{G}_4$: *This game behaves identically to the previous game, except that the adversary instantly loses (the game returns a random bit) if the following conditions hold:*

- *The adversary successfully hijacks the session run by π_P^i by a malicious* oReceive *query to π_P^i at some stage s_h prior to s^*;*
- *The adversary has not triggered* $\mathsf{Danger}(\pi_P^i.\mathsf{sid}, \mathsf{mk}^{s^*})$.

We claim that the advantage of the adversary in this game, denoted Adv_4 *is such that:*

$$\mathsf{Adv}_3 \leq max\left[\mathsf{Adv}_{\mathsf{Sign}}^{\mathsf{EUF-CMA}}(\mathcal{B}), \mathsf{Adv}^{\mathsf{GDH}}(C)\right] + \mathsf{Adv}_4,$$

for reductions $\mathcal{B}$ against the unforgeability of the signature scheme and C against the Gap-DH problem.

To understand this claim, we first note that the adversary can find itself in one of two situations: at target stage s^*, the target instance π_P^i is either the sender, or the receiver. The two situations are mutually exclusive, and as soon as the challenger guesses the target instance and stage, it will know which of those situations it is in.

Moreover, note that once successful hijacking is achieved, the instance π_P^i will no longer be partnered with its honest partner (which $\mathcal{A}$ has successfully impersonated), but rather, with the adversary itself.

We now consider the two options described above.

Suppose first that π_P^i is the receiver at stage s^*. In this case, we can construct a reduction $\mathcal{B}$ to the unforgeability of the signature scheme. This reduction generates all the private keys with the exception of sk_Q, where $Q = \pi_P^i.\mathsf{pid}$. Then the reduction simulates the game faithfully, querying its signature oracle whenever it needs a signature on behalf of Q.

In order to win its game $\mathcal{A}$ will need to query oTest *for the honest instance π_P^i at stage s^*. Suppose that $\mathcal{A}$ has not queried* oReceive *for π_P^i at stage s^*. In this case, our reduction will fail, but so will $\mathcal{A}$, since π_P^i will have no key mk^{s^*} set. Since π_P^i no longer has an honest partner, $\mathcal{A}$ cannot receive honest input from that partner. As a result, the only way for the adversary to make P ratchet to stage s^* is to produce a valid message at stage s^* (since P is the receiver at that stage), including a valid signature using sk_Q. Note also that $\mathcal{A}$ may not simply query corrupt Q for that signature key (a query that $\mathcal{B}$ would not be able to respond to), nor can it use* oAccessSK *(since either of those actions would trigger the active danger event). Hence, the adversary's only choice is to forge the signature.*

We have two situations. Either $\mathcal{A}$ does not produce that message/signature pair – in which case, both $\mathcal{A}$ and $\mathcal{B}$ lose, or $\mathcal{A}$ does produce the message/signature pair, in which case $\mathcal{B}$ can forward it, and wins.

Now we turn to the other situation, in which π_P^i is the sender at the target stage s^*. In this case, we construct an adversary C against the GDH problem that wins with at least as much probability as $\mathcal{A}$ wins its game.

The reduction C will simulate the game correctly, except that it will embed, as it elements g^a, g^b, the public identity key of Q (pk_Q) and the public ratcheting key Rchpk^{s^*} (forwarded by P at stage s^*). We note that the only way the adversary can distinguish mk^{s^*} from random is if it queried input that includes $(\mathsf{Rchpk}^{s^*})^{sk_Q}$ to the random oracle $\mathcal{RO}_2$. Note that the reduction generates all the other keys, and uses its DDH oracle to ensure consistency with respect to the challenge elements. When the adversary queries $\mathcal{RO}_2$ with an input that allows the DDH oracle to return 1, the reduction forwards that input as its guess for g^{ab}.

We have two cases. Either $\mathcal{A}$ never queries a correct input to $\mathcal{RO}_2$, in which case both $\mathcal{A}$ and C fail, or $\mathcal{A}$ does query the correct input, in which case C wins.

This gives the required bound.

Note that we have now effectively ruled out active attacks made by the adversary. We can now focus on only passive attacks.

From this point on, we will have to move through the options given in the winning conditions, starting from the target stage.

$\mathbb{G}_5$: *This game is identical to the previous one, except we substitute the key mk^{s^*} to a random (consistent) value if the adversary does* not *trigger the event $\mathsf{Danger}(\pi_P^i.\mathsf{sid}, (\mathsf{pk}_R)^{\mathsf{rchk}^{s^*}})$, where R is the receiving party in the target session at stage s^* (thus, R could be either P or its partner). In the following we show that except for breaking the GDH problem, the adversary's probability to win the two games is identical.*

Let Adv_5 be the adversary's advantage in $\mathbb{G}_5$. It holds that:

$$\mathsf{Adv}_4 \leq \mathsf{Adv}^{\mathsf{GDH}}(\mathcal{D}_0) + \mathsf{Adv}_5,$$

where $\mathcal{D}_0$ is an adversary that breaks the GDH problem.

The reduction is very similar to the second case of game $\mathbb{G}_4$ (reduction C). The adversary cannot simply leak the value of mk^{s^*} (as per the winning conditions) and it cannot perform an active attack (as per the previous game). Its only option is to query $\mathcal{RO}_2$ for the correct input that yields mk^{s^*}, including the value $(pk_R)^{\mathsf{rchk}^{s^*}}$. The reduction only has to embed its challenge tuple into that value and return it when $\mathcal{A}$ queries it to the random oracle.

As of this game, we can therefore assume that the adversary $\mathcal{A}$ triggers the event $\mathsf{Danger}(\pi_P^i.\mathsf{sid}, (\mathsf{pk}_R)^{\mathsf{rchk}^{s^*}})$ (or if that is not the case, $\mathcal{A}$ can only win with probability $\frac{1}{2}$). Note that, as per the winning conditions, the adversary cannot now also *trigger the event $\mathsf{Danger}(\pi_P^i.\mathsf{sid}, \mathsf{ck}^{s^*})$. In addition: honest instances cannot produce ratcheting keys duplicating $\mathsf{rchk}^{(x^*-(j-5),y^*)}$ (as per $\mathbb{G}_1$), and while pk_R will be used in other instances, the reduction knows the other share in the DH product. Finally, note that in order to win, the adversary* must *input a value for ck^{s^*} to $\mathcal{RO}_2$.*

$\mathbb{G}_6$-$\mathbb{G}_{5+x^*}$: *At each subsequent game hop $\mathbb{G}_{5+j}$, with $j \in \{1, \ldots, x^* - 1\}$, we move one step backwards along the x axis of the challenge stage $s^* = (x^*, y^*)$. At each step, we will replace ck^{x^*-j,y^*} to a random (consistent key if the $\mathsf{Danger}(\pi_P^i.\mathsf{sid}, (\mathsf{pk}_R)^{\mathsf{rchk}^{(x^*-\mathrm{j},y^*)}})$ event is not triggered, where j is the number of the game hop. Note that as soon as this is true for at least one ck value, the adversary's success probability turns to $\frac{1}{2}$. For each game we use the same argument as above to prove:*

$$\mathsf{Adv}_j \leq \mathsf{Adv}^{\mathsf{GDH}}(\mathcal{D}_j) + \mathsf{Adv}_{j+1}.$$

Here, $\mathcal{D}_j$ is at each time a new reduction ($j \in \{1, \ldots, x^* - 1\}$) against the GDH problem. The latter will be embedded at each time in the input to $\mathcal{RO}_2$, which is of the form: $(pk_R)^{\mathsf{rchk}^{(x^*-j,y^*)}}$.

After every individual game hop, the conclusion is the same: in order to produce the correct input to $\mathcal{RO}_2$, the adversary somehow has to have a correct value for the chain key – without triggering the danger event to that value.

$\mathbb{G}_{6+x^*}$: *At this point we have arrived at the beginning of the message chain with index y^*. We have two situations: either $y^* \geq 2$ or $y^* = 1$ (and the challenger will know at this point which situation it is in). We suppose for now that $y^* \geq 2$ (if not, the proof skips directly to the next game, ignoring this one). We split this game into two sub-hops:*

$\mathbb{G}_A$: *This game is identical to the previous, but we change ck^{1,y^*} to random if the adversary has not triggered the danger event for $(\mathsf{ipk}_R)^{\mathsf{t}_{y^*-2}}$, where R is the receiver at chain y^*. We claim that the adversary does not lose anything by this modification of the game, except if it can break the GDH problem. The reduction is similar to those above, and we lose a term $\mathsf{Adv}^{\mathsf{GDH}}(\mathcal{E}_{0,A})$. Thus:*

$$\mathsf{Adv}_{5+x^*} \leq \mathsf{Adv}_A + \mathsf{Adv}^{\mathsf{GDH}}(\mathcal{E}_{0,A}).$$

$\mathbb{G}_B$: *At this point, we replace all keys ck^{1,y^*} through to ck^{s^*} and mk^{s^*} by random, consistent values. Indeed, due to the winning-condition restrictions, at this point, the adversary is no longer allowed to trigger the danger event for $pk_S^{\mathsf{t}_{y^*-1}}$, where S is the sender at stage y^*. At this point, except by breaking the GDH assumption, $\mathcal{A}$ is no longer able to distinguish the ck^{1,y^*} from random, which, due to our previous games, will no longer allow it to distinguish any of the intermediate values from random (as it cannot endanger any of the intermediate challenge keys either). Note that at the end of this game, the adversary's winning probability will become $\frac{1}{2}$, thus ending the proof:*

$$\mathsf{Adv}_A = \frac{1}{2}.$$

$\mathbb{G}_{7+x^*}$: *The alternative to the previous game is the situation in which $y^* = 1$. We note that due to the previous games and the winning restrictions, the adversary may not at this point endanger $\mathsf{ck}^{1,1}$. Thus in game $\mathbb{G}_{7+x^*}$, we replace $\mathsf{ck}^{1,1}$ by a random, consistent value if the adversary does* not *trigger $\mathsf{Danger}((\mathsf{prepk}_R)^{\mathsf{rchk}^{0,1}})$ for a responder PR. Once more our reduction to GDH will rely on the fact that such an adversary must break the GDH assumption to distinguish $\mathsf{ck}^{1,1}$ from random. It holds that:*

$$\mathsf{Adv}_{\mathbb{G}_{5+x^*}} \leq \mathsf{Adv}_{\mathbb{G}_{7+x^*}} + \mathsf{Adv}^{\mathsf{GDH}}(\mathcal{F}).$$

$\mathbb{G}_{8+x^*}$: *This game hop consists of case-by-case reductions for the security of the master secret* ms, *similar to the analysis given by Cohn-Gordon* et al. *in Appendix C1 of the full version of their Signal-analysis paper. We notably replace this master secret by a consistent, but random value. For each case, we focus on which values are* not endangered *by the adversary (the winning conditions will not allow the adversary, by that point in the game, to know all the parts). In each case, we will lose a term equivalent to a reduction to GDH, yielding:*

$$\mathsf{Adv}_{\mathbb{G}_{7+x^*}} \leq \mathsf{Adv}_{8+x^*} + \mathsf{Adv}^{\mathsf{GDH}}(\mathcal{G}_1) + \mathsf{Adv}^{\mathsf{GDH}}(\mathcal{G}_2) + \mathsf{Adv}^{\mathsf{GDH}}(\mathcal{G}_3) + \mathsf{Adv}^{\mathsf{GDH}}(\mathcal{G}_4).$$

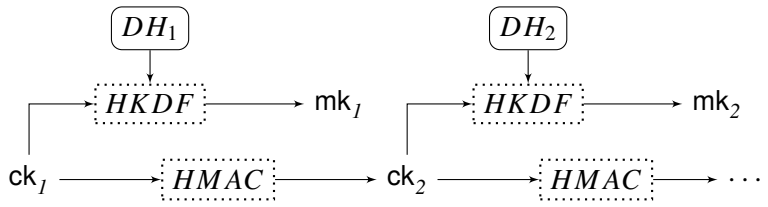
At this point we can proceed to replace all the intermediate values of ck *from the beginning of the session until the target stage* $s^* = (x^*, y^*)$ *(note that we are in the case where* $y^* = 1$*). We also replace* mk^{s^*} *by random. We argue that, due to the previous gamehops, the adversary has no means of endangering this value, and as such, we can guarantee that no such input has been made to the random oracles. At this point the adversary's advantage will be* $\frac{1}{2}$*, thus concluding the proof:* $\mathsf{Adv}_{8+x^*} = \frac{1}{2}$.

We note that when putting the full advantage inequality together, we can upper bound the value of x^* *by* $\mathsf{n_S}$.

2.4 SAMURAI

The design of MARSHAL aims at improving the healing of Signal. However, in order to handle the message-loss resilience (see Section 2.3.4.6), all the data corresponding to a given chain must be included in each message. This leads to a linear growth in the metadata inside a chain in the length of the chain. The memory content grows proportionally to the number of messages for a given chain.

We propose a variant called Signal-like Asynchronous Messaging with Message-loss resilience, Ultimate healing and Robust against Active Impersonations (SAMURAI) with a modification on the key derivation inside a chain (compare to MARSHAL). Informally, we split the derivation of the chain keys ck from the message keys mk. This allows to compute all the ck as in Signal, *i.e.*, the ck are derived from symmetric ratchet, while the mk are computed via asymmetric ratchet. Hence, the key derivation of a given is the balance between symmetric and asymmetric ratcheting taking the best of both algorithms. We can illustrate this hybrid ratcheting as:



2.4.1 Description of SAMURAI

We now describe in details our protocol SAMURAI: Signal-like Asynchronous Messaging with Message-loss resilience, Ultimate healing and Robust against Active Impersonations.

2.4.1.1 Registration

Parties using SAMURAI must first register by generating key pairs and uploading the public part to the server. For party P to register, it uploads a long-term identity key ipk_P, a medium-term key prepk_P (signed with the identity key ik_P), multiple ephemeral pre-keys ephpk_P, multiple medium-term stage keys T_0. The latter key is a novelty toward Signal, it will be used to initiate a communication by instantiating the first chain. An additional key pair is

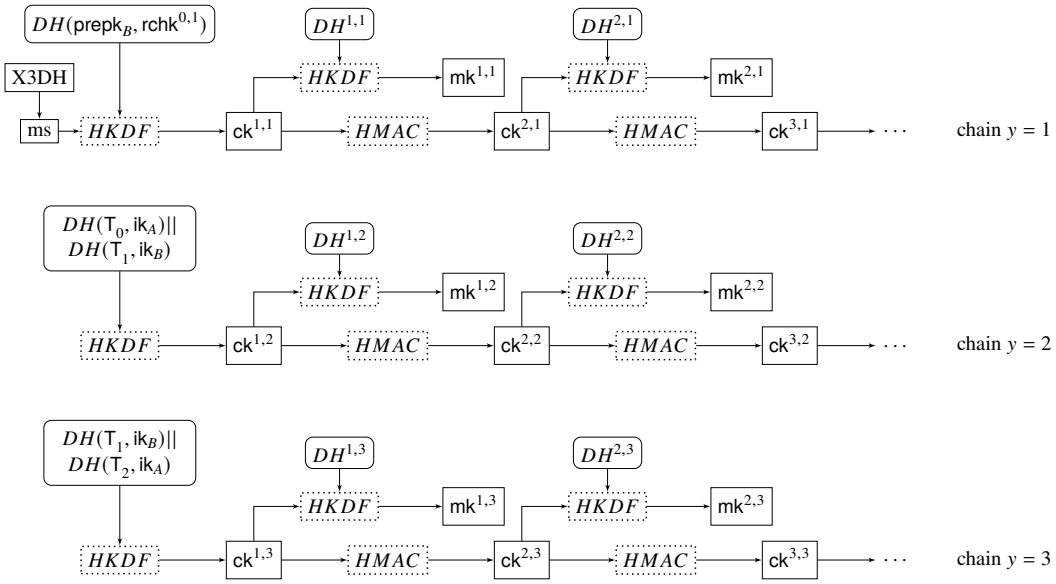


Figure 2.16 SAMURAI *key schedule.*

also generated (but not uploaded to the server), a long-term signature keys $(\mathsf{SKM}_P, \mathsf{PKM}_P)$. These keys will be used throughout the communication and appended in the associated data in each partner's first respective chain of messages.

2.4.1.2 Setup

If Alice wishes to speak with Bob (both registered as described in the above paragraph), she needs to run a setup phase similar to Signal.

Computing the Master Secret ms**.** As in Signal, the shared secret is computed with X3DH algorithm [MP16b]. In details, when Alice wants to initiate a session with Bob, she queries the server for Bob's following values: the identity key ipk_B, a signed pre-key ephpk_B (if any), and a medium-term key T_B^0, denoted T_0 for simplicity. Upon receiving those keys, A generates its own ephemeral key ek_A. Finally, the master secret is computed as:

$$ms = \mathsf{prepk}_B^{\mathsf{ik}_A} || \mathsf{ipk}_B^{\mathsf{ek}_A} || \mathsf{prepk}_B^{\mathsf{ek}_A} || \mathsf{ephpk}_B{}^{\mathsf{ek}_A}$$

Initialising the first chain. Alice randomly generates a ratchet key pair $(\mathsf{rchk}^{0,1}, \mathsf{Rchpk}^{0,1})$. She computes a DH value using its ratchet key and prepk_B passed as input (along with ms) to a key derivation function to produce the first chain key $\mathsf{ck}^{1,1}$.

2.4.1.3 Communication

Our protocol SAMURAI heals faster than Signal thanks to its perpetual asymmetric ratcheting occurring at each stage. Hence, for stage $(1, 1)$, Alice needs ratcheting information from Bob which is given by the public key T_0 generated during registration. Then, for each starting chain stage $(1, m)$ with $m > 1$, the party initiating the chain generates a random value t_m with public part T_m. The latter is sent as part of associated data of all messages for chain m and will be used to initiate chain $m + 1$.

Inside a chain, *i.e.*, at stage (ℓ, m) for $\ell > 1$, the current speaker generates key ratchet $(\mathsf{rchk}^{\ell,m}, \mathsf{Rchpk}^{\ell,m})$ used to derive message key. The crucial point of SAMURAI is that two derivations coexist within a given chain. The chain keys $\mathsf{ck}^{\ell,m}$ are derived as in Signal, *i.e.*, $\mathsf{ck}^{\ell,m} = \mathsf{HMAC}(\mathsf{ck}^{\ell-1,m})$, while the message keys are derived only when the message is received by the communication partner, *i.e.*, $\mathsf{mk}^{\ell,m} = \mathsf{HKDF}(\mathsf{ck}^{\ell,m}, \sigma_{\ell,m}||(\mathsf{Rchpk}^{\ell,m})^{\mathsf{ik}_B})$. Thus deriving the chain key is independent of the values mk (but the inverse is not true though).

The notations we use are the followings:

$$DH^{x,y} := \begin{cases} \sigma_{x,y}||DH(\mathsf{ipk}_B, \mathsf{rchk}^{x,y}), \text{ for } y \text{ odd} \\ \sigma_{x,y}||DH(\mathsf{ipk}_A, \mathsf{rchk}^{x,y}), \text{ for } y \text{ even} \end{cases}$$

with

$$\sigma_{x,y} := \begin{cases} \mathtt{SIGN}_{sk_A}\left(\mathsf{T}_{y-1}||\mathsf{Rchpk}^{x,y}\right), \text{ for } y \text{ odd} \\ \\ \mathtt{SIGN}_{sk_B}\left(\mathsf{T}_{y-1}||\mathsf{Rchpk}^{x,y}\right), \text{ for } y \text{ even} \end{cases}$$

The value σ is a signature on elements that are used to derived the message key. It ensures the persistent authentication (as introduced in [BBB+19]) of the values that are part of the evolution of the message keys. The aim of such signature is to minimize the possibility of an attacker to hijack a session by injecting its own key material. Indeed, the PCS property allows to regularly randomize key derivation for communicating partners which can "eject" a passive attacker but an active attacker could also use this feature to inject its own values. From this observation, we can infer that authentication toward key evolution is critical which, we believe, is ensured by this value σ.

We detail the first rounds of communication in Figure 2.16 and 2.17.

Alice's first message. At session setup, Alice has generated its cross-user ratchet keys $(\mathsf{t}_1, \mathsf{T}_1)$, and computed the chain key $\mathsf{ck}^{1,1}$. Now she generates the same-user ratchet key $\mathsf{rchk}^{1,1}$ and computes the corresponding public part $\mathsf{Rchpk}^{1,1} = g^{\mathsf{rchk}^{1,1}}$. The message and chain-keys are computed as follows:

$$\mathsf{ck}^{2,1} \leftarrow \mathsf{HMAC}(\mathsf{ck}^{1,1})$$
$$\mathsf{mk}^{1,1} \leftarrow \mathsf{HKDF}(\mathsf{ck}^{1,1}, DH^{1,1})$$

In chains $y = 1$ and $y = 2$, both communicating partners are including in their messages some metadata that are used for session setup and also specific to the session. The metadata for $\mathsf{AD}_{y=1}$ thus includes the public identity keys of Alice and Bob, the medium-term and ephemeral keys of Bob as recovered by Alice from server, the value T_0 from the server, Alice's ephemeral public key used in the computation of the master secret, and two of Alice's ratchet public keys: its first same-user ratchet key $\mathsf{Rchpk}^{0,1}$, and its first cross-user ratchet key t_1.

There are also stage-specific metadata (which are included in each message, especially for chain $y > 2$). Those information includes: the stage index $(1, 1)$ and the same-user ratcheting public key $\mathsf{Rchpk}^{1,1}$.

Alice $(\mathsf{ik}_A, \mathsf{ipk}_B, \mathsf{prepk}_B, \mathsf{ephpk}_B, \boxed{\mathsf{T}_0})$		**Bob** $(\mathsf{ik}_B, \mathsf{ipk}_A, \mathsf{prek}_B, \mathsf{ephk}_B, \boxed{\mathsf{T}_0})$
Session initialization: initiator Alice, responder Bob.		
$\mathsf{ek}_A, \mathsf{rchk}^{0,1}, \boxed{\mathsf{t}_1, \mathsf{rchk}^{1,1}} \xleftarrow{\$} \mathbb{Z}_q$; $\mathsf{T}_1 = g^{t_1}$; $\mathsf{Epk}_A = g^{\mathsf{ek}_A}$; $\mathsf{Rchpk}^{0,1} = g^{\mathsf{rchk}^{0,1}}$; $\mathsf{Rchpk}^{1,1} = g^{\mathsf{rchk}^{1,1}}$ $ms = \mathsf{prepk}_B^{\mathsf{ik}_A} \| \mathsf{ipk}_B^{\mathsf{ek}_A} \| \mathsf{prepk}_B^{\mathsf{ek}_A} \| \mathsf{ephpk}_B{}^{\mathsf{ek}_A}$ $\mathsf{ck}^{1,1} = \mathsf{HKDF}(\mathsf{prepk}_B^{\mathsf{rchk}^{0,1}} \| ms)$ $\mathsf{ck}^{2,1} = \mathsf{HMAC}(\mathsf{ck}^{1,1})$ $\mathsf{mk}^{1,1} = \mathsf{HKDF}(\mathsf{ck}^{1,1}, \boxed{\sigma_{1,1} \| (\mathsf{ipk}_B)^{\mathsf{rchk}^{1,1}}})$		
First message: stage $(1,1)$, Alice is the sender, Bob, the receiver.		
$\boxed{AD_{y=1}} = \mathsf{Epk}_A \| \mathsf{ipk}_A \| \mathsf{ipk}_B \| \mathsf{prepk}_B \| \mathsf{ephpk}_B \| \mathsf{T}_0 \| \mathsf{Rchpk}^{0,1} \| \mathsf{T}_1$ $\boxed{AD_{1,1}} = (1,1) \| \mathsf{Rchpk}^{1,1} \| \sigma_{1,1}$ $c_{1,1} = \mathtt{AEAD.Enc}_{\mathsf{mk}^{1,1}}(M_{1,1}; AD_1 \| AD_{1,1})$	$\xrightarrow{c_{1,1}, \mathtt{SIGN}_{sk_A}(c_{1,1}),\ pk_A, \mathtt{SIGN}_{\mathsf{ik}_A}(pk_A)}$	Verify signature on pk_A and $\sigma_{1,1}$ $ms = \mathsf{ipk}_A^{\mathsf{prek}_B} \| \mathsf{Epk}_A^{\mathsf{ik}_B} \| \mathsf{Epk}_A^{\mathsf{prek}_B} \| \mathsf{Epk}_A^{\mathsf{ephk}_B}$ $\mathsf{ck}^{1,1} = \mathsf{HKDF}((\mathsf{Rchpk}^{0,1})^{\mathsf{prek}_B} \| ms)$ $\mathsf{ck}^{2,1} = \mathsf{HMAC}(\mathsf{ck}^{1,1})$ $\mathsf{mk}^{1,1} = \mathsf{HKDF}(\mathsf{ck}^{1,1}, \sigma_{1,1} \| (\mathsf{Rchpk}^{1,1})^{\mathsf{ik}_B})$ $M_{1,1} = \mathtt{AEAD.Dec}_{\mathsf{mk}^{1,1}}(c_{1,1})$.
ℓ-th **message**: stage $(\ell, 1)$, Alice is the sender, Bob, the receiver.		
$\mathsf{rchk}^{\ell,1} \xleftarrow{\$} \mathbb{Z}_q$, set $\mathsf{Rchpk}^{\ell,1} = g^{\mathsf{rchk}^{\ell,1}}$ $\mathsf{ck}^{\ell+1,1} = \mathsf{HMAC}(\mathsf{ck}^{\ell,1})$ $\mathsf{mk}^{\ell,1} = \mathsf{HKDF}(\mathsf{ck}^{\ell,1}, \sigma_{\ell,1} \| \mathsf{ipk}_B^{\mathsf{rchk}^{\ell,1}})$ $\boxed{AD_{\ell,1}} = (\ell,1) \| \mathsf{Rchpk}^{\ell,1} \| \sigma_{\ell,1}$ $c_{\ell,1} = \mathtt{AEAD.Enc}_{\mathsf{mk}^{\ell,1}}(M_{\ell,1}; AD_1 \| AD_{\ell,1})$	$\xrightarrow{c_{\ell,1}, \mathtt{SIGN}_{pk_A}(c_{\ell,1}),\ pk_A, \mathtt{SIGN}_{\mathsf{ik}_A}(pk_A)}$	Verify leftover signatures $\mathsf{ck}^{\ell+1,1} = \mathsf{HMAC}(\mathsf{ck}^{\ell,1})$ $\mathsf{mk}^{\ell,1} = \mathsf{HKDF}(\mathsf{ck}^{\ell,1}, \sigma_{\ell,1} \| (\mathsf{Rchpk}^{\ell,1})^{\mathsf{ik}_B})$ $M_{\ell,1} = \mathtt{AEAD.Dec}_{\mathsf{mk}^{\ell,1}}(c_{\ell,1})$.
Switching speakers: Bob comes online and begins a new ratcheting chain.		
		$\boxed{\mathsf{t}_2, \mathsf{rchk}^{1,2} \xleftarrow{\$} \mathbb{Z}_q; \mathsf{T}_2 = g^{t_2}, \mathsf{Rchpk}^{1,2} = g^{\mathsf{rchk}^{1,2}}}$ $\mathsf{ck}^{1,2} = \mathsf{HKDF}(\boxed{\mathsf{T}_1^{\mathsf{ik}_B} \| \mathsf{ipk}_A{}^{t_0}})$ $\mathsf{ck}^{2,2} = \mathsf{HMAC}(\mathsf{ck}^{1,2})$ $\mathsf{mk}^{1,2} = \mathsf{HKDF}(\mathsf{ck}^{1,2}, \boxed{\sigma_{1,2} \| (\mathsf{ipk}_A)^{\mathsf{rchk}^{1,2}}})$
Bob's message, stage $(1,2)$: Bob is the sender, Alice is the receiver.		
		$\boxed{AD_{y=2}} = \mathsf{T}_2$ $\boxed{AD_{1,2}} = (1,2) \| \mathsf{Rchpk}^{1,2} \| \sigma_{1,2}$
	$\xleftarrow{c_{1,2}, \mathtt{SIGN}_{pk_B}(c_{1,2}),\ pk_B, \mathtt{SIGN}_{\mathsf{ik}_B}(pk_B)}$	$c_{1,2} = \mathtt{AEAD.Enc}_{\mathsf{mk}^{1,2}}(M_{1,2}; AD_2 \| AD_{1,2})$
Verify signature on pk_B and $\sigma_{1,2}$ $\mathsf{ck}^{1,2} = \mathsf{HKDF}((\mathsf{ipk}_B)^{t_1} \| (\mathsf{T}_0)^{\mathsf{ik}_A})$ $\mathsf{ck}^{2,2} = \mathsf{HMAC}(\mathsf{ck}^{1,2})$ $\mathsf{mk}^{1,2} = \mathsf{HKDF}(\mathsf{ck}^{1,2}, \sigma_{1,2} \| (\mathsf{Rchpk}^{1,2})^{\mathsf{ik}_A})$ $M_{1,2} = \mathtt{AEAD.Dec}_{\mathsf{mk}^{1,2}}(c_{1,2})$		

Figure 2.17 SAMURAI *protocol execution between Alice and Bob for the first few stages. The grey boxes indicate modifications with respect to Signal protocol [CGCD+17]. The transmitted data is also different and not in grey for more clarity.*

Finally, when Alice wants to send her first message, she computes the authenticated encryption:

$$c_{1,1} = \mathtt{AEAD.Enc}_{\mathsf{mk}^{1,1}}(M_{1,1}; \mathsf{AD}_{y=1} \| \mathsf{AD}_{1,1})$$

and sends: $c_{1,1}$, a signature on it, Alice's public signature key pk_A, and a signature on it.

Alice's $(\ell, 1)$ **message,** $\ell > 1$**.** Alice has computed $\mathsf{t}_1, \mathsf{T}_1$, $\mathsf{AD}_{y=1}$, and the key $\mathsf{ck}^{\ell,1}$; she

now generates new same-user ratcheting key-pair $(\mathsf{rchk}^{\ell,1}, \mathsf{Rchpk}^{\ell,1})$. The key update relies on both long-term keys, for persistent authentication, and this same-user ratcheting key, for healing:

$$\mathsf{ck}^{\ell+1,1} \leftarrow \mathsf{HMAC}(\mathsf{ck}^{\ell,1})$$
$$\mathsf{mk}^{\ell,1} \leftarrow \mathsf{HKDF}(\mathsf{ck}^{\ell,1}, DH^{x,y})$$

The stage-specific metadata consists of the stage $(\ell, 1)$ and the ratcheting key $\mathsf{Rchpk}^{\ell,1}$. Then Alice computes $c_{\ell,1}$ and sends: the ciphertext, a signature on it, its signature public key, and a signature on that.

Note that this procedure applies to *all* messages (ℓ, m) for $\ell > 1$ and $m \geq 1$, in replacing the y stage-index above, from 1 to m.

Decryption (Bob side). Bob needs to first compute the setup value as Alice did, namely the master secret ms and the chain key $\mathsf{ck}^{1,1}$. For this, Bob asks the server for Alice's identity key and verifies if it corresponds to the one given in $\mathsf{AD}_{y=1}$. If the latter verification is valid, Bob also checks the signature on pk_A, and, if the verification returns 1, it stores that key as A's signature key. With all the public values Bob has recovered, he can compute ms (as Alice did), the chain key and the first message key using the metadata of the first message. With the latter, Bob can decrypt the authenticated message.

Bob's first message. B generates a new cross-user ratcheting value along with its public part $(\mathsf{t}_2, \mathsf{T}_2)$ and a same-user ratcheting key-pair $(\mathsf{rchk}^{1,2}, \mathsf{Rchpk}^{1,2})$. Bob computes: $\mathsf{ck}^{1,2} \leftarrow \mathsf{HKDF}\left((T_1)^{\mathsf{ik}_B} || \mathsf{ipk}_A^{t_0}\right)$, then its first sending keys:

$$\mathsf{ck}^{2,2} \leftarrow \mathsf{HMAC}(\mathsf{ck}^{1,2})$$
$$\mathsf{mk}^{1,2} \leftarrow \mathsf{HKDF}(\mathsf{ck}^{1,2}, DH^{1,2})$$

Then analogously to Alice side, Bob splits the metadata into the two auxiliary values $\mathsf{AD}_{y=2}$ and $\mathsf{AD}_{1,2}$. The signed public key pk_B is also appended to each of the messages in stages with chain-index $y = 2$, *cf.* Figure 2.17.

2.4.2 Security Analysis

Since MARSHAL and SAMURAI are equivalent (in terms of PCS), we do not consider a security analysis in the same framework for both. To avoid repetitions in the security analysis of SAMURAI we chose the model given in the next chapter dedicated to analyse protocol in the PCS angle. Thus, the security analysis of SAMURAI is done in Section 3.3.2.2 providing a wider analysis for our variants.

2.5 Implementation

We have implemented both MARSHAL and SAMURAI to evaluate their performances toward Signal. Since the level of PCS is improved by using additional tools (*e.g.*, Diffie-Hellman

value, signature), we need to ensure that the overhead remains acceptable in practice. Our implementation is a proof-of-concept rather than a fully integrated solution to MARSHAL and SAMURAI; indeed our implementation covers registration and messaging phase but we exclude out-of-order message decryption.

The performance evaluation is done via two aspects, the execution time and the memory consumption. The three protocols are implemented in Java based on the initial implementation of Signal [15]. We used low-level libsignal functionalities but had to re-implement its more abstract layers, to fit with MARSHAL and SAMURAI. The libsignal library uses interfaces in order to store and recover keys and session state data; it allows us to abstract the central server and simulate exchanges between two parties within the same process. Our implementation uses similar elliptic curve material than the one for Signal. For signatures, we use XEd25519 on Curve25519. We use similar authenticated encryption algorithms relying on AES with 128-bit keys. However, while Signal uses CBC mode and an hmac256-MAC, we prefer to use AES-GCM with a 12-byte IV and a 16-byte tag.

We have divided our analysis in three experiments:

- **Session Setup:** A run-time experiment to evaluate the time for starting a session.
- **Scenario:** This experiment tries to replicate a usual communication between two users.
- **Same Chain:** We analyse both run-time and memory in this experiment where new messages are derived inside a given chain.

2.5.1 Session Setup and Scenario run-time

All the results indicated in this section are the mean result over 1000 executions of each given test. See Table 2.1 for the results. We only consider run-time in those two experiments since this is the most noticeable metric in those cases. We do analyse the memory evaluation but in the Same Chain experiment where the memory parameter renders a crucial parameter.

Session setup. The first differences between Signal, MARSHAL and SAMURAI occur during registration and session setup. For the setup, Alice and Bob need to generate two new elements: a signature key and a Diffie-Hellman public value. While the master secret of both sessions is similar, the first chain and message keys are generated differently in Signal and MARSHAL/SAMURAI. Our Session-Setup tests covers: key-generation steps of both Alice and Bob, publishing a PreKeyBundle, initiating a session that uses that bundle, including the encryption and decryption of a first message.

Scenario. This experiment gives a trend for the protocols in a *regular* communication context. The goal is to evaluate the average run-time for a practical example. The scenario experiment includes the session setup, the first message of Alice, a first reply of Bob (thus initialising a new chain), then 3 messages of Alice, and finally 3 of Bob.

Interpretation. We notice in Table 2.1 that MARSHAL and SAMURAI take twice as long as Signal for the session setup test. This is partly because we also require the use of signatures

[15] Available at `https://github.com/signalapp/libsignal-protocol-java`

Test	Signal	MARSHAL	SAMURAI
Session Setup	4.26	6.42	6.31
Scenario	6.74	21.6	21.6

Table 2.1 *Average run-time for each test in ms.*

in our variant. Note that the signature scheme used is the same as Signal; however, a faster signature scheme could significantly improve performance. We also note that, while the relative increase in run-time is significant, the absolute measurement is still low in both tests. Notice also that MARSHAL and SAMURAI are alike in those experiments as expected: SAMURAI improves the performances in terms of memory toward MARSHAL but not the run-time. The next test, Same Chain, is dedicated to differentiate MARSHAL and SAMURAI.

2.5.2 Same Chain experiment

The last experiment concerns the run-time and memory management in a given chain. We increase the number of messages (up to 600) inside a chain to characterize each protocol. We give the results in Figure 2.18 and Figure 2.19.

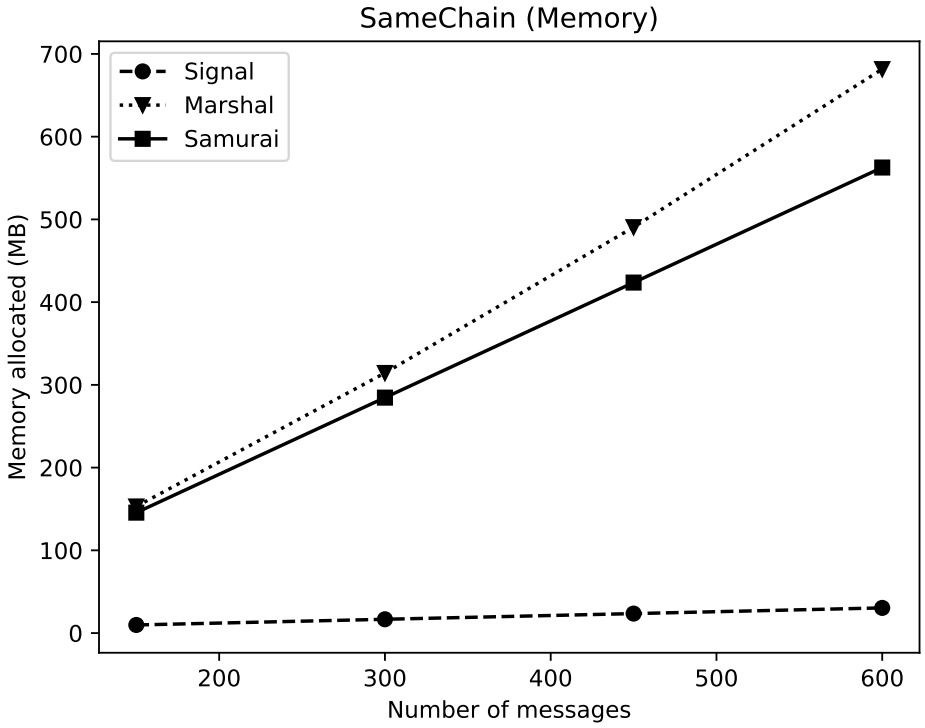


Figure 2.18 *Average memory consumption depending on the number of messages in a given chain.*

From the results, we can see that MARSHAL is the most memory consuming protocol. This comes from the out-of-order message feature where metadata from all the previous messages (inside the chain) need to be included within a new message. The performance of SAMURAI is better than MARSHAL yet still far from Signal.

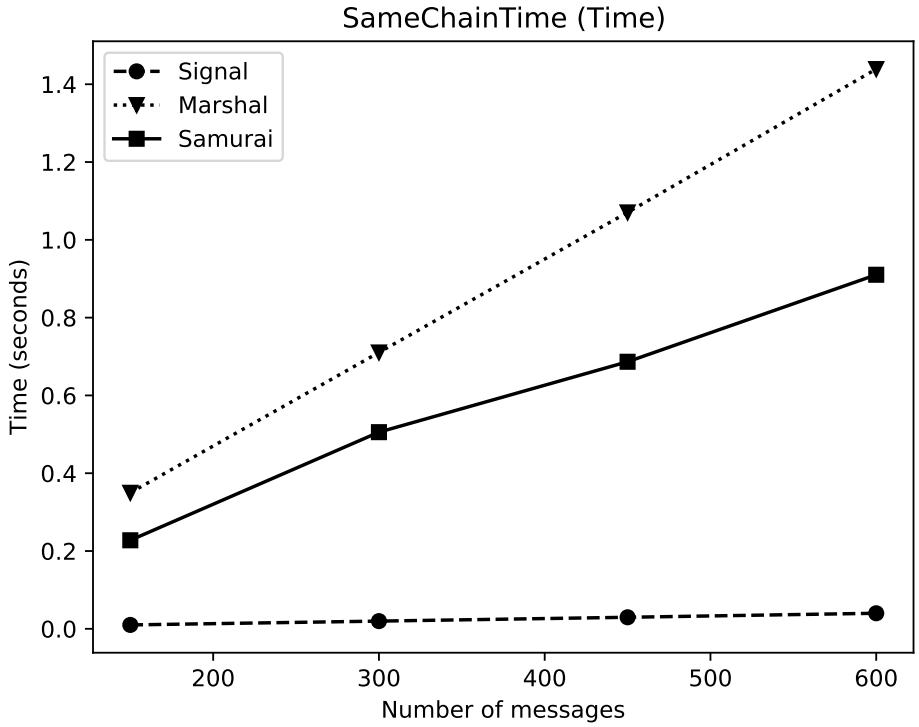


Figure 2.19 *Average execution time depending on the number of messages in a given chain.*

2.6 Conclusion

We propose two variants of the widely used asynchronous messaging protocol Signal. With MARSHAL and SAMURAI, we achieve better security at comparatively little cost. Unlike alternative approaches to designing ratcheted key-exchange, which follow a modular design (typically based on KEMs), we try to stick close to Signal's original structure, thus showing how to achieve better post-compromise security.

Our protocols depart from the key observation that Signal's comparative lack of PCS is due to the frequency of asymmetric ratchets and lack of persistent authentication. The latter is fixed by adding long-term keys at every new stage. The former is dealt with by adding asymmetric ratchets *at every stage*. To do so, we require a long-term key stored on the semi-trusted Signal server, and we ensure that message-loss resilient is achieved by providing the a list of correct ratcheting keys at every stage of a given chain.

The main difference between MARSHAL and SAMURAI does not lie in the security but rather on the performance side. A feature we want to keep (for practical issue) is the out-of-order message property. In MARSHAL, a stage, inside a chain, is derived using all the data from previous stages thus auxiliary information grows linearly when continuing a chain. The fix of SAMURAI allows to remove this dependence with a constant size of auxiliary information while keeping the out-of-order feature.

Finally, we have implemented our protocols to evaluate the practical cost of our modifications. Our implementation does not require fundamental changes to the basic cryptographic primitives used in Signal. In addition, experiments show that significant benefits to post-compromise security our protocol brings do not come at a too-significant cost, the run-times of our ratchets and message-exchanges remaining under 10 ms mark in Java implementation.

Chapter 3

Measuring the PCS Healing

Numerous messaging protocols have arisen with their own specifities and security properties. We propose a way to compare them in a generic framework for a specific feature: the Post-Compromise Security. To show the validity of our model, we first analyse Signal. Then we show that comparison is possible by applying it to two variants of Signal, SAID and SAMURAI. Finally, we show the expressive aspect of our framework by taking a use case in the 5G network procedures.

Contents

3.1 Introduction

In the previous chapter, we introduced two variants enhancing the *Post-Compromise Security* (PCS) of Signal messaging protocol. The latter was used as a basis for group messaging schemes, such as ART [CCG+18] and MLS [BBR+22]; 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 [BGB04], Matrix [mat19], Wire [Gmb21], also guarantee some measures of PCS. The rise of asynchronous messaging protocols comes after Snowden's revelations, who exposed the reality of mass surveillance by security agencies such as the NSA or GCHQ. 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 the PCS [CCG16]. 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.

Cohn-Gordon *et al.* [CGCD$^+$17] 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.

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?

While protocols with PCS can be analysed independently, a natural question arises when we want to find the "best" PCS-protocol: can we express the speed of healing given a generic framework? From this, there are two crucial points to investigate, the quantification of healing speed (*i.e.*, its metric) and a model for underlying protocols.

Our first approach is naturally directed toward secure channel establishment schemes, *e.g.*, asynchronous messaging protocol like Signal [CGCD$^+$17], since the setup can be easily modulable for the latter. Indeed, we can have only two parties, Alice and Bob (further substituted to a client and a server), with a clear evolution of the states define as stages (see Figure 3.1).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. An intuitive way of quantifying the healing speed from this model is to simply count the messages loss after a compromise, corresponding to the number of stages that are compromised.

Our goal is not to propose a survey on each protocol featuring PCS notion but a precise framework and why it is working. For this, we take Signal protocol which is well analysed and accepted by the community for many years. Yet, for showing the expressiveness of our model, we add a variant of Signal, SAID, further allowing the comparison between the two protocols. Those two protocols being relatively equivalent (considering all types of secure channel establishment protocols), we also add the 5G handovers suites to widen our framework and show its full expressiveness.

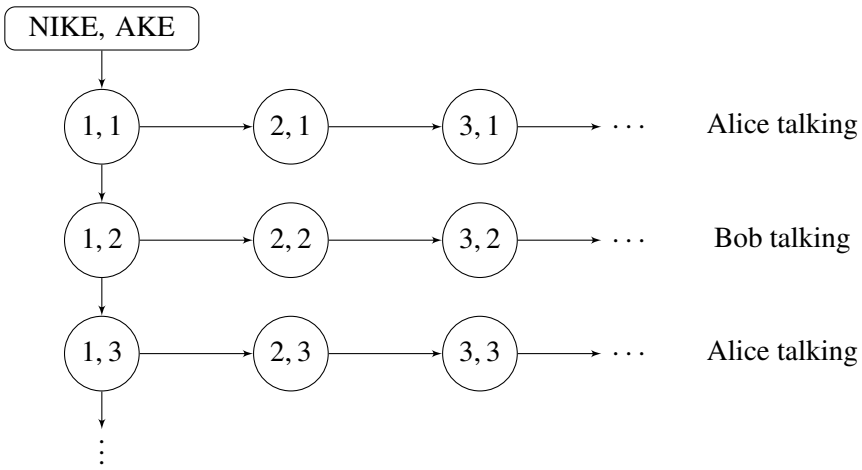


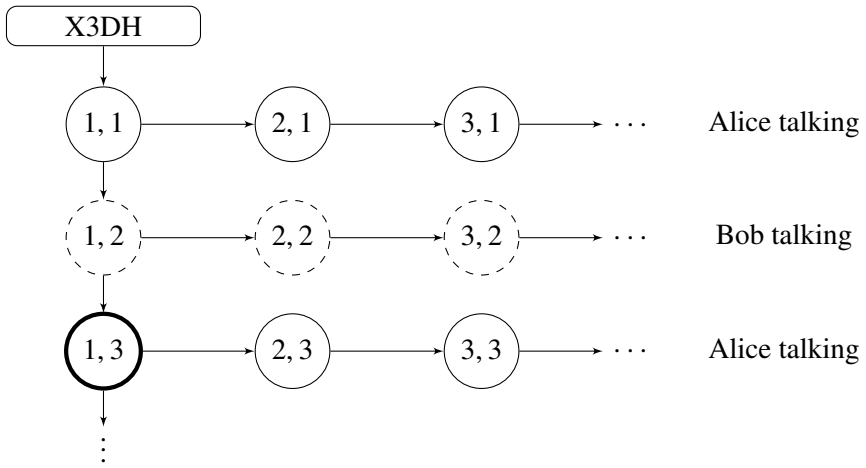
Figure 3.1 *Stages for secure channel establishement. The NIKE (Non-Interactive Key Exchange) and AKE (Authenticated Key Exchange) are sub-protocols used to derive shared secrets. The latter are not part of our analysis.*

3.1.1 Example

Suppose that an attacker manages to compromised a single (but full) chain of stages while being ejected from the communication at the beginning of the new chain (Figure 3.2a). During that compromise, depending on its type, the attacker will learn a subset of private keys which it might, or might not depending on type, be able to use in an active attack. Our metric measures the *number of messages* required, per message-chain, for the security of the channel to heal after this corruption. Thus, in this example, Signal is $(\infty, 1)$-PCS resistant with respect to some class of adversaries because the honest parties lose channel security for at most ∞ 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 (*i.e.*, the protocol heals at the latest at stage $(1, y^* + 1)$, having lost potentially the confidentiality of all the messages in chain y^* with $x \geq x^*$).

Blazy *et al.* [BBB+19] showed how to improve the guarantees provided by Signal, by reducing the duration before the scheme heals and making it harder for Big Brother to impersonate the two parties. However, it seems difficult to compare the two schemes directly. Blazy *et al.*'s protocol (called SAID) is identity-based, whereas Signal is not. The former makes use of a secure element to store some long-term credentials, whilst the latter does not. At a high level, both protocols guarantee post-compromise security – so are they perhaps equivalent?

(a) *Toy example for Signal*



(b) *Toy example for SAID.*

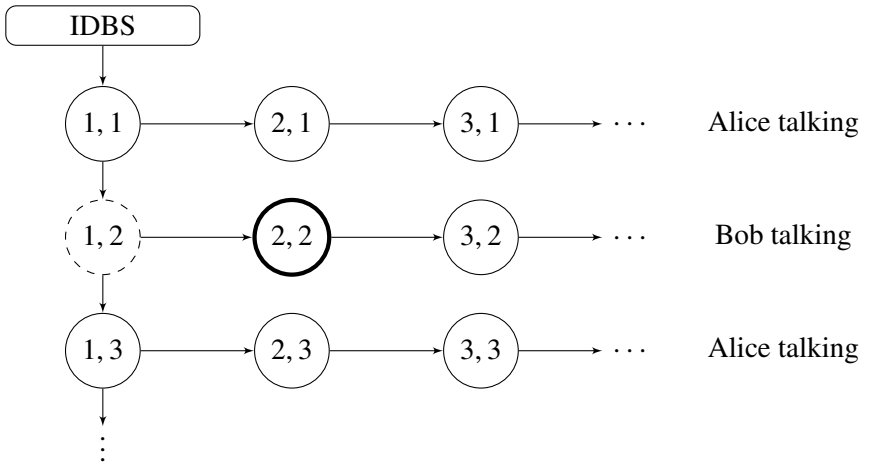


Figure 3.2 *Example of compromise and healing for Signal and SAID. The dashed circles represent compromised stages while thick circles represent healed stage.*

In the example of Figure 3.2b, there is one compromised stage and an immediate healing occurring the next stage. In this example, there is a $(1, 0)$-PCS security for SAID which is an optimal healing toward the given compromise (recall that analysing PCS security depends on a compromise so there is always a stage lost).

From the toy examples of Figure 3.2, a natural question arises: is SAID better than Signal for PCS? Looking at the numbers and the answer is clearly yes but those protocols are not equivalent and the answer becomes uneasy to tell. Indeed, SAID is an identity-based protocol with a *Key Distributor Center* (versus a semi-honest server for Signal) and a specific key hierarchy. One goal for our metric is to give generic framework to make this comparison possible and fair.

3.1.2 Goals for our metric

We want to formally define a *post-compromise security (PCS) metric* for a broad class of protocols, which we call SCEKE schemes (for Secure-Channel Establishment schemes with Key-Evolution). Crucially, SCEKE protocols allow key material to evolve over time, with

channel keys being used only for a short duration of time (*i.e.*, a *stage*).

The goal of PCS adversaries against SCEKE protocols is to learn something about a target channel (also called message) key, following a compromise of that party or its partner. During that compromise, depending on its type, the attacker will learn a subset of private keys which it might, or might not depending on type, be able to use in an active attack. Our metric measures the *number of messages* required, per message-chain, for the security of the channel to heal after this corruption. Optimal healing corresponds to $(1, 0)$-PCS security, while the worst healing is (∞, ∞)-PCS security, *i.e.*, the protocol's security never heals. Those extreme cases are quite straightforward but our model becomes necessary for cases in between.

Apart from quantifying the speed of recovery, the goal is also to provide a taxonomy of adversaries, classified by 3 sets of 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 within memory). A very strong adversary might be a trusted party (like Signal's credential server), able to actively hijack sessions and to fully compromise all the data belonging to a party.

To showcase the broad reach of our metric, we use it to compare 4 schemes that would otherwise be hard to compare: the PKI-based Signal asynchronous messaging protocol (analyzed by Cohn-Gordon *et al.* [CGCD$^+$17]), the identity-based SAID asynchronous-messaging protocol [BBB$^+$19], our own variant of Signal built specifically toward PCS, SAMURAI; and the 5G handover protocols in mobile networks. For the latter protocol, there is no attempt (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 in order to obtain much faster healing.

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[1]. 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,

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

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 [CCG+18, BBR+22], 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.

3.1.3 Related work

Our work is the first (to our knowledge) to quantify PCS-security in terms of healing speed and adversaries ranked by their access, degree of activity, and power. However, we note that exact provable-security approaches in asynchronous messaging (such as the analyses we use in this paper for Signal [CGCD+17] and SAID [BBB+19]) already take an important step in comparing the security of protocols, by precisely formalizing attackers and their winning conditions. Our main added contributions here, however, are: generalizing a framework that goes beyond just asynchronous messaging in the shape of SCEKE protocols (we instantiate SCEKE using 5G handover protocols); providing a taxonomy of adversarial types in terms of their power, access, and reach; and thus creating a metric that can compare instantiations of secure-channel establishment protocols with key evolution.

Although comparatively infrequent, taxonomies and metrics have already been introduced in cryptography. An eminent example is the taxonomy of privacy notions by Pfitzmann and Hansen [PH10], which ranks and classifies subtly-different terms referring to user privacy (such as anonymity, privacy, unlinkability, undetectability, etc.). Unlike them, we focus on the privacy of information (channel security) and narrow the scope of our analysis down to a specific type of protocol; 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 [PH10] due partly because of its generality.

Our methodology better resembles taxonomy efforts such as [FG94, DLL12], 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 formal-methods approach.

The specificities of our model resemble those of Fischlin and Günther [FG14], which extend prior work by Bellare and Rogaway [BR93a, CK01]; however, we consider security with respect to the complementary property of Post-Compromise Security, and also consider stage evolution in two directions (horizontal and vertical). Finally, our metric allows the quantification of the speed of healing, rather than the secure/insecure label typically produced

in the asymptotic case of the left-of-right indistinguishability game in AKE. A parallel line of work recently introduced by Brzuska *et al.* [BCK22] analyzes, in a compositional framework (using state-separating proofs [BDF+18]), the security of the multi-party asynchronous messaging protocol MLS. Unlike that technique, ours is not composable[2], but 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 four protocols (with apparently incomparable degrees of PCS) to analyze: Signal, SAID, SAMURAI and the 5G Handover protocols. Our instantiation of SCEKE protocols only feature two endpoints, and we thus do not treat, for instance, the case of multi-party asynchronous messaging and/or multiple groups as is done by Cremers *et al.* [CHK21].

That said, our result can apply when considering the security of group asynchronous messaging instantiated via pairwise channels. Our framework also captures sequential ratchets of the type described in the literature of ratcheted key-exchange protocols, a line of work begun by Bellare *et al.* [BSJ+17] and continued through numerous publications.

3.1.4 Outline

We give the formal definition of SCEKE protocol in Section 3.2. This generic protocol is constructed to model protocols in order to be able to compare them. For this define SCEKE protocol in 3.2.1, the adversary in 3.2.2 (*e.g.*, the available oracles for $\mathcal{A}$), the class of adversaries in 3.2.3 (defining a fine-grained access to sequence of queries), and finally our metric in 3.2.4.

Then, in Section 3.3, the plausibility of our model is checked using the analysis of already provably secure protocols (*e.g.*, Signal); the analysis is extended to less studied protocol 5G handovers in 3.3.4. We present the latter procedures which are then fitted to our model; this analysis brings a solution to improve the protocol.

Finally, we conclude in Section 3.4 by interpreting the results given by our model.

3.2 Our PCS metric for SCEKE protocols

Our taxonomy of adversaries and quantification of healing apply to a generalization of two-party secure-channel establishment protocols that feature key-evolution. Although post-compromise security usually refers to secure channels that are long-lived, such as those featured in asynchronous messaging, we note that this can be expanded to scenarios in which specific secure-channels are short lived, but then evolve into secure channels whose keys were derived from past sessions.

3.2.1 Definition of SCEKE Protocols

We formally characterize *secure-channel establishment with key-evolution (SCEKE) protocols*. Such schemes allow parties Alice and Bob to initially establish a secure channel (by

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

agreeing on some initial key-material), and then to maintain that secure channel over a long period of time by sequential evolutions of the key material to ensure two basic properties:

PFS: We want to guarantee that, if a user or an instance are fully corrupted at a given moment, that keys established before that corruption are still secure;

PCS: In addition, if a user or instance are fully corrupted at some moment, then through evolution, the security of future keys in that same, long session will be once more guaranteed after a given interval.

Our metric will measure the minimal interval needed for security to return, with respect to several classes of adversaries (which we present in Section 3.2.3). We begin by describing a general syntax for SCEKE protocols. The latter are composed of parties (which are not equivalent) evolving in sessions through specific proceedings (*i.e.*, stages).

Parties and super-users. We consider protocols featuring two types of participants: parties P belonging to some set $\mathcal{P}$, and a super-user $\hat{S}$, which is usually an entity playing a special part (like a PKI-providing centralized server used in Signal to store user public keys, or the key-derivation center present in identity-based infrastructures). Regular parties can and do run protocol sessions in order to establish a secure channel with key evolution, while the super-user does not.

Prior to the system coming online, a setup algorithm is run by one or several parties. We associate super-user $\hat{S}$ with a pair of private and public keys $(\hat{S}.\mathsf{sk}, \hat{S}.\mathsf{pk})$, and each party P with private/public identity-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 $\perp$). The public keys are assumed to exist *prior* to the user registering to the system.

After setup, the next step is user-registration, at which point each party P will register with the super-user. At this point, $\hat{S}$ begins to build a database of entries, indexed by party identifiers P, which must be unique per party. The contents of the entries are different per protocol, and indeed play a crucial part in how post-compromise secure a protocol is with respect to a specific, powerful type of adversary – notably an insider attacker as we formalize below.

Sessions, instances, and stages. Once they have registered with the super-user, parties can then run protocol sessions with each other. Following typical authenticated key-exchange formalizations, a protocol *session* happens between two party-*instances*. The i-th instance of P is denoted π_P^i.

From the point of view of the two parties, each protocol session has three types of steps: an initialisation step, which only occurs once per protocol instance (at the beginning); a recurring sending step, which happens every time the instance processes and *sends* a message to its partner; and a recurring receiving step, which corresponds to an instance *receiving* – and processing – a message from its partner. The last two types of actions depend on a notion that is crucial to our framework, that of *stage*.

We associate stages to a variable (protocol-specific) number of messages, during which the parties store an unevolving, specific *message key*. When that key evolves, we have moved on to the next stage. We index stages by a tuple (x, y) with $x, y \geq 1$, such that the first

communication stage is $(1, 1)$. From this point onward, encrypted communications will use the same key $\mathsf{mk}^{1,1}$, until that key evolves into a new key, through either *horizontal* or *vertical* evolution, as follows:

Horizontal evolution: Stage (x, y) turns into $(x + 1, y)$, we increment the stage's x coordinate, but not its y coordinate. This is the weaker of the two ways to transform keys.

Vertical evolution: Stage (x, y) turns into $(1, y + 1)$, we increment the stage's y coordinate and "reset" its x coordinate to 1. This is the stronger of the two ways to transform keys.

The vagueness of our definitions of stages is intentional, since we aim to capture a generic type of key-evolution. As a single rule, however, honest parties can never evolve "backwards" (*e.g.*, send a message at stage (x, y) and then again at $(x - 1, y)$ or at $(\cdot, y - 1)$). We depict the evolution across stages in Figure 3.1.

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

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

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

stages: a list with elements (s, v), with stages $s = (x, y)$ and values $v \in \{0, 1\}$ for which 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 for stage s describing all data sent or received for this stage. We denote $\pi_P^i.T[s]$.

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.

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

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

$\mathtt{aKeyGen}(1^\lambda) \rightarrow (\mathsf{ik}, \mathsf{ipk})$ *: this algorithm is run by a party P to output public/private long-term credentials* $(\mathsf{ik}, \mathsf{ipk})$*. Either of those keys could be a special symbol* $\perp$*. These keys will be used in practice to authenticate the party when registering (and perhaps also during the protocol).*

$\Pi_{\mathtt{UReg}}(P, \hat{S}) \rightarrow (\{\mathsf{sk}, \mathsf{pk}\}, b)$ *: this interactive protocol is run by a party P with the super-user* $\hat{S}$*. The latter will output a bit b (if the user is registered correctly, that bit will be* 1*), while the former will output public/private credentials* $(\mathsf{sk}, \mathsf{pk})$ *for P. The super-user may decide to add some or all of the entries to its database.*

$\Pi_{\mathtt{Start}}(P, \mathsf{role}, \mathsf{pid}, \hat{S}) \rightarrow (\pi_P^i, b)$ *: it is run interactively between P and the super-user* $\hat{S}$*, for the purpose of creating an instance of P that will purportedly be talking to an instance of* pid*, such that P plays a given* $\mathsf{role} \in \{I, R\}$*, either initiator or responder. In the event of a success,* $\hat{S}$ *outputs b, while P outputs a handle* π_P^i *on its i-*th *instance.*

Some initial key material might be already set up during this phase (such as a master secret).

$\mathtt{aSend}(\pi_P^i, s, M, AD) \rightarrow (\pi_P^i, C, AD^*) \cup \perp$ *: the inputs are a state instance π_P^i, a stage s, a message M, associated data AD, and outputs the same instance with updated state π_P^i, a ciphertext C, associated data AD^* or a special symbol $\perp$.*

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

$\mathtt{aRGen}(1^\lambda) \rightarrow (\mathsf{rchk}, \mathsf{Rchpk})$ *: outputs a public/private key-pair used to refresh keys. We use the typical denomination of* ratchet keys *for these, although our concept here is more general, as we state no specific purpose for the resulting keys. Either one or both output keys could be a special symbol $\perp$.*

Before defining the correctness of communication protocol, we give the definition of matching conversation. This ensures that the communicating party of some user is indeed the intended partner.

Definition 19 (Matching conversation) *Let SCEKE be a SCEKE protocol, and A, B two users of instances π_A^i and π_B^j respectively. π_A^i and π_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* $\pi_B^j.\mathsf{pid} = A$.

Definition 20 (Correctness) *If π_A^i and π_B^j have matching conversation, then a SCEKE protocol SCEKE is* correct *if both conditions holds:*

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

3.2.2 Adversarial model

Our adversary is a Probabilistic Polynomial-Time adversary $\mathcal{A}$, which manipulates honest parties by means of *oracles*. We present in the next section a taxonomy of adversaries based on several characteristics – depending on their type, such attackers could have access to only a subset of the oracles we present 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:** These are keys that (intentionally) repeat in at least two sessions[3]. More formally, a key k is cross-session if there exist distinct instances π_P^i, π_P^j of registered party P, and 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 keys ik_P and sk (identity and registration keys) belonging to P are included in this category. We denote the set of cross-session keys of party P as $P.X\mathsf{sid}$.

[3]We thus formally exclude collisions in randomness

- **Cross-stage Keys:** These are keys that (intentionally) repeat in at least two stages of the same session, but do not repeat in two distinct sessions. More formally, there exists an instance π_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 π_P^i as $\pi_P^i.X\mathsf{stage}$.
- **Stage-specific Keys:** These keys only occur in one stage of one protocol instance π_P^i, in other words: $k \in \pi_P^i.\mathsf{var}[s]$ for some stage s, but $k \notin (P.X\mathsf{sid} \bigcup \pi_P^i.X\mathsf{stage})$. We denote by $\pi_P^i.1\mathsf{stage}$ the set of all stage-specific keys of instance π_P^i.

Oracle access. The adversary can register malicious users, corrupt users to get any of the three types of keys included above (through three separate oracles), and manipulate communication by instantiating new sessions and sending/receiving messages. The adversary ultimate goal is to learn at least one bit about a target message key that is freshly and honestly generated. Thus, each session needs to also store the following 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 precise game played by the adversary in Section 3.2.4, and a number of interesting adversarial *types* in Section 3.2.3. However, in all these games, the adversary has access to (a subset of) the following oracles:

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

$\mathsf{oStart}(P, \mathsf{role}, \mathsf{pid}, hon)$: 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 $\hat{S}$.

$\mathsf{oTest}_b(\pi_P^i, s)$: for honest parties, valid instances, valid stages and correctly generated session key, returns the true message 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.

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

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

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

$\mathsf{oReveal.XStage}(\pi_P^i, s)$: 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.

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

Like in the model used for SAID [BBB+19], $\mathcal{A}$ does not have access to the real ciphertext, which can help to distinguish the message keys.

3.2.3 A taxonomy of adversaries

We classify adversaries in terms of 3 criteria: reach, power, and access. Although attackers play more or less the same security game and have the same security goals, they might have access to more or less private values belonging to honest parties, and they might be allowed different sequences of oracle queries. Our classification of adversaries is the composition of the 3 criteria defined below. The classes of adversaries are illustrate in Figure 3.3

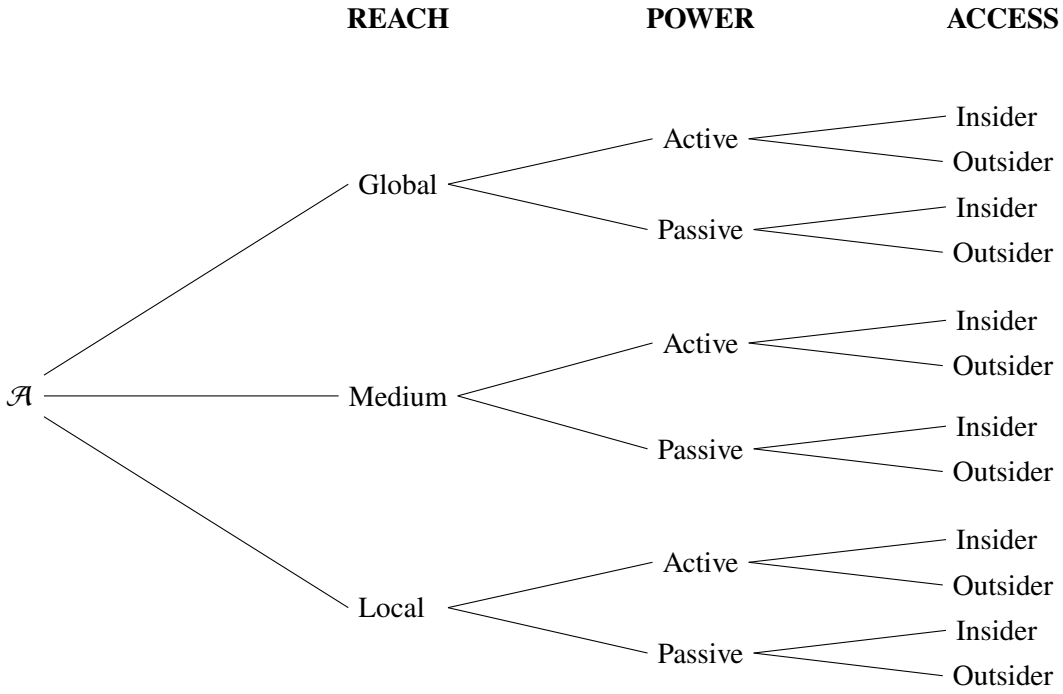


Figure 3.3 *Taxonomy of adversary.*

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:

- Local adversaries: are only allowed access to the oReveal.1Stage oracle;
- Medium adversaries: may query both oReveal.1Stage and oReveal.XStage, but not oReveal.XSid;
- Global adversaries: 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 π_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 π_A^i and its partner π_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 $v \notin \pi_A^i.\mathsf{var}[s_h] \cup \pi_B^i.\mathsf{var}[s_h]$.

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 Figure 3.4, we recap the adversary's access to oracle depending on its type.

3.2.4 A metric for PCS

The adversary $\mathcal{A}$ plays against a challenger C in the following security game $\mathsf{Exp}^{PCS}_{\mathsf{SCEKE}}(\lambda, \mathcal{A})$, which is also depicted in Figure 3.5:

- C runs `aSetup` and forwards all the public values to $\mathcal{A}$. 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 Figure 3.4):
  - $\mathsf{oUReg}(P)$ (all attackers);

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	✓	✓	✓	✓	✓	✓

Figure 3.4 *Available oracles depending on type, labelled* reach||power||access. *For instance, LAO denotes Local Active Outsider adversary. We omit* oTest, oUReg *and* oStart *oracles since all adversaries may query them. H denotes an honest call to the oracle.*

- oStart$(P, \mathsf{role}, \mathsf{pid}, 1)$ (outsider $\mathcal{A}$) and oStart$(P, \mathsf{role}, \mathsf{pid}, 0)$ (for insiders);
- oSend$(\pi_P^i, s, \perp)$ (passive $\mathcal{A}$) and oSend(π_P^i, s, AD) (active $\mathcal{A}$);
- oReceive$(\pi_P^i, s, \perp)$ (passive $\mathcal{A}$) and oReceive(π_P^i, s, AD) (active $\mathcal{A}$);
- oReveal.XSid(P) (global $\mathcal{A}$);
- oReveal.XStage(π_P^i, s) (medium $\mathcal{A}$);
- oReveal.1Stage(π_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 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.

$\underline{\mathsf{Exp}^{PCS}_{\mathsf{SCEKE}}(\lambda, \mathcal{A})}$

$(\hat{S}.\mathsf{sk}, \hat{S}.\mathsf{pk}, \mathsf{pparam}) \leftarrow C^{\mathtt{aSetup}(1^\lambda)}$

$(\mathcal{P} = \{P_1, \cdots P_{\mathsf{n}_\mathcal{P}}\}) \leftarrow C(\lambda, \mathsf{n}_\mathcal{P})$

$(\mathsf{ik}_i, \mathsf{ipk}_i) \leftarrow C^{\mathtt{aKeyGen}(1^\lambda)} \quad \forall i \in \{1, \cdots, \mathsf{n}_\mathcal{P}\}$

$O_{\mathsf{type}} \leftarrow \left\{ \begin{array}{l} \mathsf{oUReg}(\cdot), \mathsf{oStart}(\cdot,\cdot,\cdot,\cdot), \mathsf{oReveal}[\mathcal{A}.\mathsf{reach}](\cdot,\cdot), \mathsf{oSend}(\cdot,\cdot,\cdot,\cdot), \\ \mathsf{oReceive}(\cdot,\cdot,\cdot,\cdot), \mathcal{RO}_1(\cdot), \mathcal{RO}_2(\cdot) \end{array} \right\};$

$(\pi_P^\star, s^\star) \leftarrow \mathcal{A}^{O_{\mathsf{type}}}(1^\lambda)$

$K \leftarrow \mathsf{oTest}_{b^\star}(\pi_P^\star, s^\star)$

$d \leftarrow \mathcal{A}^{O_{\mathsf{type}}}(\lambda, \mathsf{n}_\mathcal{P}, K)$

$\mathcal{A}$ wins iff. $d = b^\star$ and $(\neg\mathsf{oUReg}(P) \vee \neg\mathsf{oUReg}(\pi_P^i.\mathsf{pid})) = \top$

Figure 3.5 *The PCS game* $\mathsf{Exp}^{PCS}_{SCEKE}(\lambda, \mathcal{A})$ *between adversary* $\mathcal{A}$ *and challenger* C*, parametrized by the security parameter* λ *and number of* honest *parties* $\mathsf{n}_\mathcal{P}$*.* $\mathcal{A}$ *can query a set of oracles* O_{type}*, subject to type. We denote by* oReveal$[\mathcal{A}.\mathsf{reach}]$ *the precise reveal oracle allowed to* $\mathcal{A}$*, subject to its reach (local, medium, or global).*

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.

Definition 21 ((χ, Υ)-PCS security) *A SCEKE protocol is (χ, Υ)-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 π^i_P, the last stage for which $\mathcal{A}$ queried* oReveal.XStage *or* oReveal.1Stage *for either π^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_{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 negligible advantage to win if* oTest *is queried for s_{Test} other than those specified in the first bullet point.*

We allow both χ and Υ to take a special value ∞, which corresponds to "an arbitrary number of stages" obtained through horizontal and respectively through vertical evolution.

3.3 Use-cases of our metric

We apply our metric to 4 use cases: the PKI-based messaging protocol Signal, the Identity-Based messaging protocol SAID, our own variant SAMURAI (or equivalently MARSHAL) 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.

Since our first idea was to being able to compare variants of Signal toward PCS, secure asynchronous-messaging protocols are natural instantiations of SCEKE. To prevent Person-in-the-Middle attacks, Signal users register their data on a centralized credential server, which will be our super-user $\hat{S}$. By contrast, the SAID protocol relies on identity-based cryptography, run with a Key-Distribution Center. Our SCEKE framework will allow us to consider the security of the protocol with respect to these powerful insiders, a fact often overlooked by prior analyses [CGCD+17, BBB+19].

Another aspect of protocols like Signal and SAID that is modelled in the SCEKE framework is the evolution from one stage to another. Asynchronous-messaging protocols are essentially turn-based conversations. The initiator of the session, say Alice, uses Bob's

Keys	Signal	SAMURAI	SAID	*5G-SCEKE*'s
Cross-Session	ik,prek	ik,prek,SKM	ik, ID.sk,IBS.sk	K
Cross-Stage	rk,rchk, tmp	T	ms, rk,rchk	K_{AMF} = rk
Single-Stage	ephk, ms, mk, ck	ephk, ms, mk, ck, rchk	mk, ck, r	rchk =v , K_{AS} = mk, K_{gNB} = ck

Table 3.1 *Taxonomy on keys used in Signal, SAID and our 5G handover procedures model. The grey box indicates additional key of* 5G-SCEKE$^+$ *compared to* 5G-SCEKE

credentials and some randomness to calculate initial key material. She is the first to speak, at stage $(1,1)$, sending one encrypted message only. For as long as Alice is sending messages, there is no randomness added by Bob (*i.e.*, the evolution of stages is horizontal). When Bob replies, he will include its own key material and randomness into the conversation (*i.e.*, vertical evolution of stages). A crucial point here is that healing is possible only when the speaker is changing (*e.g.*, Bob replies). However, for SAMURAI case, the randomness is added in each step so there is no need to wait for one partner to reply. This property needs to be carefully handled in the model.

The major strength of our model comes from the ability to compare protocols. This is possible by giving classes of elements for which they are equivalent. Since the key material is closely related to PCS notion (*i.e.*, what consequences for which keys) and protocols (*i.e.*, the key hierarchy), we need to categorize each key falling into the taxonomy of keys we gave above. We summarize the key material in Table 3.1 which will be detailed further depending on the protocol.

Finally, before giving all the details of the use-cases, we show the result produced by our model in Table 3.6. We give an interpretation of our results after analysing them in our framework.

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

Figure 3.6 *Results for our metric on PCS-security for Signal,* SAMURAI*, SAID, 5G handover and its variant denoted* 5G-SCEKE$^+$.

3.3.1 The Signal protocol

Signal is a natural instantiation of SCEKE protocols. Like most asynchronous-messaging schemes, conversations on this medium are turned-based. The two speakers, who need not be online simultaneously, will alternate messages. Each message corresponds to one stage precisely (*i.e.*, each message key is used only once). There are two types of evolutions, at any given point in the conversation: either the same person who was speaking will send a new message (corresponding to a horizontal evolution), or the speaker will change (thus, a vertical evolution).

Signal also features a natural super-user in the form of a centralized credential server storing user public keys. Our SCEKE framework will allow us to consider the security of the protocol with respect to this powerful insider, a fact often overlooked by prior analyses [CGCD+17, BBB+19].

We begin this section by showing how to model Signal as a SCEKE protocol. Then we quantify its PCS-security with respect to all the adversaries we described in Section 3.2.3.

3.3.1.1 Signal description as SCEKE

We consider a set $\mathcal{P}$ of users that need to be registered before they can use Signal. Our super-user $\hat{S}$ is a centralized PKI server.

Setup. During the global setup of the protocol, a number of algorithms are chosen by $\hat{S}$, including a signature algorithm whose keys are public Diffie-Hellman keys, the hash functions and KDFs used, and a secure-channel establishment protocol to be used at registration (such as TLS). Simultaneously, the super-user generate a (certified) pair of private/public long-term keys $(\hat{S}.\mathsf{sk}, \hat{S}.\mathsf{pk})$ which it uses in order to establish secure channels with the users (these could, for instance, be signature keys to be used in a TLS-DHE handshake like that included in TLS 1.3). We do not make any assumptions with respect to the type of keys considered, but we assume that honest super-users generate strong key material. All the chosen algorithms 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. We assume that the super-user and parties P run a secure-channel establishment protocol that minimally allows for the authentication of both parties. Subsequently, each user P registers a key-bundle consisting of: a long-term identity key ipk_P, a medium-term key prepk_P signed with ik_P, and optional ephemeral *public* keys ephpk_P. Both ipk_P and prepk_P are used across multiple sessions, whereas each ephemeral public key ephpk_P is only used in one session and then removed from the server. We stress that the server is never given the user's private keys – in that sense, it is only semi-trusted (believed to be honest-but-curious).

Instance initialisation. In order to communicate with each other, both the initiator and the responder need to create session instances (which will hopefully then have matching conversation). Say that Alice (the initiator) wants to begin a session with Bob. She begins

by querying the semi-trusted server, over an authenticated channel, for Bob's credentials, which allows Alice to establish the master secret as the concatenation of Diffie-Hellman products:

$$\mathsf{ms} := (\mathsf{prepk}_B)^{\mathsf{ik}_A} || (\mathsf{ipk}_B)^{\mathsf{ek}_A} || (\mathsf{prepk}_B)^{\mathsf{ek}_A} || (\mathsf{ephpk}_B)^{\mathsf{ek}_A}$$

using newly-generated randomness ek_A. Alice will also use the randomness-generating algorithm aRGen to output a ratcheting key-pair $(\mathsf{rchk}^1, \mathsf{Rchpk}^1)$.

The master secret will yield an intermediate *root key* 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. Alice will include as metadata to that message the public key Epk_A, the identities of Alice and Bob, an identifier of the keys prepk_B and ephpk_B that she used, and the ratcheting public key Rchpk^1. This metadata will pass as Associated (Authenticated) Data (AAD) to the AEAD ciphertext.

We note that Bob cannot respond to Alice in this session until she has sent at least one message to him; hence, the message at stage $(1, 1)$ is always sent by the initiator of the session to its responder.

Bob will also need to do the same computations as Alice. In order to produce his instance, he will use an authenticated channel to retrieve Alice's information from $\hat{S}$. Then, Bob uses his private keys prek_B and ephk_B, as well as the metadata appended in Alice's message, to compute the master secret and retrace the rest of Alice's computations. We recall that stages for which Alice will be speaking are indexed $(\cdot, y)$ with an odd y. The stages for which Bob is the message-sender are indexed $(\cdot, y)$ for even y.

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. Each stage corresponds to a single encrypted message. In each message, the included metadata allows the receiver to make his keys evolve, either horizontally (he receives and decrypts a new message) or vertically (the receiver decides to starts talking).

The cryptographic material will evolve as follows:

- **Symmetric ratcheting:** In Signal, when the sender at stage (x, y) wants to send a new message (for stage $(x + 1, y)$), symmetric ratcheting occurs. Recall that during key derivation, the chain key $\mathsf{ck}^{x,y}$ will yield both the message-key at stage (x, y) *and* the next chain key $\mathsf{ck}^{x+1,y}$. The symmetric ratchet consists of feeding $\mathsf{ck}^{x+1,y}$ to the KDF to obtain $\mathsf{ck}^{x+2,y}$ and $\mathsf{mk}^{x+1,y}$.
- **Asymmetric ratcheting:** When the message-sender changes (the stage evolves from (x, y) to $(1, y + 1)$), an asymmetric ratchet occurs. At this point, the new sender introduces new randomness into the conversation in the form of a public ratchet key Rchpk^{y+1}; this value will be used, together with the former sender's last ratchet key Rchpk^y, in a Diffie-Hellman product to form a value we denote as $\mathsf{DH}^{0,y+1} = DH(\mathsf{Rchpk}^y, \mathsf{Rchpk}^{y+1}) = (\mathsf{Rchpk}^y)^{\mathsf{rchk}^{y+1}}$. The exception to this rule is stage $(0, 1)$, for which $\mathsf{DH}^{0,1} = DH(\mathsf{Rchpk}^1, \mathsf{prepk}_B)$, assuming Bob is the session's responder. The value DH^{y+1} is used to compute a temporary, chain-specific secret, which is either

a root key (for odd $y+1$) or a temporary key tmp^{y+1} (for even $y+1$). The latter keys are fed into a KDF in order to derive $\mathsf{ck}^{1,y+1}$.

Recall that Figure 2.4 gives this key-derivation process.

Each Signal stage (x, y) is associated with metadata, consisting of the identities of the two speakers, the ratchet public key Rchpk^y, and the index x of the message. Exceptionally, for messages sent at stages $(\cdot, 1)$, the metadata must also include the public key Epk_A corresponding to Alice's private key ek_A used during session initialisation. This metadata is sent as Associated Authenticated Data (AAD) within each AEAD-encrypted ciphertext[4]. Thus, this data passes in clear, but is authenticated as part of the ciphertext.

Signal with acknowledgements. More recent implementations of Signal have slightly evolved from the core protocol we described in this thesis, 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 [DH21].

Comparing security models. Our framework can be seen, in many ways, as a generalization of Cohn-Gordon *et al.*'s Signal-specific security model [CGCD+17]. 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 [CGCD+17] 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 Figure 3.7, we can infer that:

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

Thus the adversaries captured in [CGCD+17] 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 described above are in fact also found amongst the winning conditions of [CGCD+17], signifying that the same

Yet, in reality (as described in the proofs), the winning predicates of [CGCD+17] imply that the adversary does not essentially benefit from the additional freedom given by those

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

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

Figure 3.7 *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 [CGCD+17]. 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.*, $x > 1$).*

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.

3.3.1.2 The PCS-security of Signal

Key-material. We begin the analysis by categorizing the key material that goes into the key-derivation of Signal, depending on whether it is stage-specific, cross-stage, or cross-session.

The keys used in Signal for a single stage only are: the message[5] keys $\mathsf{mk}^{x,y}$, the chain keys $\mathsf{ck}^{x,y}$, the master secret ms, 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 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-stage keys. We give a summary of the key material in Table 3.1.

Theorem 2 *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 (∞,∞)-PCS secure.*

Note that the results are also systematized in Figure 3.6. The proofs of this theorem consist of two types of statements: first, we need to show an attack for the stages that are

[5]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.

vulnerable to the attacker, then we need to prove that beyond those stages, security holds. The second parts of the proofs can be found afterwards, but we briefly indicate the attacks providing the first part of the proofs below.

Local Passive Outsider We claim that in this case, *the protocol is* $(\infty, 1)$*-PCS-secure*. Assume that the adversary's last oReveal.1Stage query is at stage (x, y); although the channel will be insecure for all the messages in chain y that follow after stage (x, y), the channel will heal at stage $(1, y + 1)$.

The security loss is a result of the symmetric ratchets: once $\mathsf{ck}^{x,y}$ is compromised, $\mathcal{A}$ will learn all the chain and message keys derived symmetrically from it. On the other hand, the ratchet key rchk^y is not amongst the information 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 By using its additional oReveal.XStage oracle, the adversary can now also learn ratchet keys rchk, and root/temporary keys. We claim that Signal is $(\infty, 2)$*-PCS-secure* against this type of attacker.

This is mainly because knowledge of the ratchet key 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 $y + 1$. Fortunately, this will stop at stage $(1, y + 2)$, since $\mathcal{A}$ cannot use rchk^y to compute $\mathsf{DH}^{0,y+2}$, thus giving our bound. We note that this is one of the main results in [CGCD+17].

Global Passive Outsider With the attacker having now additional access to oReveal.XSid oracle, it can obtain user identity keys and pre-keys. However, these values cannot help a passive adversary beyond learning the master secret ms, which it can learn anyway by querying the oReveal.1Stage oracle. Thus, Signal is $(\infty, 2)$*-PCS-secure* against this type of attacker.

Local Active Outsiders We begin by handling this weakest form of active outsider attacks. Recall that the difference between passive and active attackers is that the latter can actively *use* the information it captures through corruption.

Consider now an attacker with only access to oReveal.1Stage, having performed its last reveal query at stage (x, y). With the key ck at hand, $\mathcal{A}$ can now generate correct chain and message keys for the remainder of chain y, and send its own messages to the receiver of that chain, change the order of the messages, etc. This power, however, stops at stage $(1, y + 1)$, because the adversary has no access to either root, or temporary keys, which are fed into the beginning of the key derivation at each chain. This indicates $(\infty, 1)$-PCS security.

A much more insidious attack is the following: the adversary queries oReveal.1Stage at stage $(1, y)$ for some y, *before* the sending party for chain y. It chooses its own ratchet key for chain y, which we denote $\hat{\mathsf{Rchpk}}$ and inserts that into the metadata of a ciphertext that it sends the receiver at stage $(1, y)$ using the keys it has just recovered. At this point, the true sender and the true receiver will have different ratchet keys for chain y, and they will no longer have matching conversation. Fortunately, the attacker will not be able to truly insert itself in the conversation – again because it is missing the root/temporary keys. Thus, although it is a Denial of Service (DoS) attack impacting the PCS-security within chain y,

thus yielding $(\infty, 1)$-PCS security.

Other Active Outsiders For medium and global active adversaries, 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 now turn the DoS into a full hijack: it inserts $\mathsf{Rch}\hat{\mathsf{pk}}$ towards the receiver, and then uses the root key it learns via oReveal.XStage to keep up with all future ratchets from now on. This compromises all the future keys in these sessions, yielding an (∞, ∞)-PCS security.

Insider Passive Attacks In the case of a passive attacker, knowledge of the super-user's private key $\hat{S}.\mathsf{sk}$ 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 insider active 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 (∞, ∞)-PCS security (no healing at all) for all insider active attackers.

Security proofs. We now give the security proofs of Signal for each adversary's type. Those proofs aim at providing an upper bound (while the attacks previously presented were lower bound) of our metric.

We assume that all KDFs are modelled as random oracles. Each key k has $|\mathbf{k}|$ possible values. Note that the key space might be of same size for all keys (*e.g.*, $|\mathbf{k}|$, 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 n_π created by any given party.

The proofs are organized through game hops where the first game is the original security game (see Figure 3.5 of Sec. 3.2.4).

$\mathbb{G}_0$: This game corresponds to the original security game (Figure 3.5 of Sec. 3.2.4). The advantage of $\mathcal{A}$ against this game is Adv_0.

$\mathbb{G}_1$: In this game 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 oTest.

If $\mathcal{A}$ queries another parties, session or tested stage then 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 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 ms is guaranteed. Indeed, ms is computed using ik and also ephk thus by uniqueness of the former and the latter, we have uniqueness of the shared secret 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 rchk. We have:

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

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

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

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

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

For clarity, we keep the notation prek_Q instead of prek_Q^i (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 rk_1. Note that the value ms (the second input of the KDF) can be learned by any reach's adversary. [6]

$\mathbb{G}_6$: In this game the challenger accepts collision of ik_P and 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:

[6] ms is a single stage key so any local, medium or global adversary can reveal it.

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

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 Figure 3.6), 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 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 prek_Q respectively. Notice that $\mathcal{A}$ cannot query the $\mathsf{oReveal.1Stage}$ oracle on 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 used in other sessions), it must ensure a valid simulation for those (non-tested) stages. In either cases, $\mathcal{B}$ randomly chooses the value 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 ϵ_{GDH} the advantage of $\mathcal{B}$ solving the GDH problem, we have:

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

$\mathbb{G}_8$: This game ensures the indistinguishability of the rk/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 $\mathsf{n}_{\mathsf{y-max}}$:

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

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 rk/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 ck or 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 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 (χ, Υ) 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 oReveal.1Stage oracle. This leads to compromising the full chain y because of the symmetric ratchet deriving the keys (both ck and mk). The adversary has then compromised $(\infty, 1)$ stages but no more because the next chain is initialised with cross-stage keys (*i.e.*, rchk).

Medium Passive Outsider. In this case, the adversary can reveal single and cross stages keys (via the oReveal.XStage oracle) in a passive way, and it has no information on the server-stored keys. Recall that (cf Figure 3.6), 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 ms). The keys prepk_Q and $\mathsf{Rchpk}_P^{0,1}$ are now in the adversary's reach possibility, via query to 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:

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

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 rk/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 rk_1 to get all the needed information to derive the keys of chain 2. It can also derive the 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 $\mathrm{CDH}(\mathsf{rchk}^{0,3}, \mathsf{rchk}^{0,2})$ 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 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:

$$\mathsf{Adv}_6 \leq \mathsf{Adv}_7 + \epsilon_{EUF-CMA}$$

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

Medium/Global Active Outsider. For those two adversaries, Signal is (∞, ∞)-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 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 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 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 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 (∞, ∞)-PCS secure protocol for active insider adversary.

3.3.2 SAMURAI protocol

SAMURAI uses Signal backbone structure while improving PCS. The infrastructure in both protocols are identical: two speakers exchange messages initialised through a super-user storing the public keys. Notice that, as in Signal, the server is never given the private keys of users thus considered semi-trusted (in the honest-but-curious model).

We start by showing how to model SAMURAI as a SCEKE protocol, then we quantify its PCS healing rate with respect to our taxonomy of adversary (defined in Section 3.2.3). Recall that our metric is determined in two steps: first we show an attack thus finding a lower bound then we show, via a security proof, an upper bound meaning that the communication

is secure again (if so).

3.3.2.1 SAMURAI as a SCEKE

The description of SAMURAI is close to the one of Signal. The main changes concern the key material. We give a quick description while further details are given in Section 3.3.1 and also in Section 2.4.1 for the full description of SAMURAI. We consider a set $\mathcal{P}$ of users that need to be registered before they can use Signal. Our super-user $\hat{S}$ is a centralized PKI server.

Setup. This phase constitutes the setup of $\hat{S}$ in terms of algorithm routines that will be used during use. For instance, $\hat{S}$ setups hash functions, KDFs, signature scheme. It is also generating a key pair $(\hat{S}.\mathsf{sk}, \hat{S}.\mathsf{pk})$ which it uses in order to establish secure channels with the users. All the chosen algorithms 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, a signature key pair $(\mathsf{SKM}_P, \mathsf{PKM}_P)$ used for authentication during the communication. Each party also uploads a cross-stages key to the server denoted T_P^0 (later simply denoted T_0). The latter will be used to initiate the first chain.

Instance initialisation. In order to securely communicate, Alice and Bob, need to compute a master secret ms following the X3DH protocol. If Alice initiates the communication, she retrieves the public keys of Bob from the semi-trusted server and computes:

$$\mathsf{ms} := (\mathsf{prepk}_B)^{\mathsf{ik}_A} || (\mathsf{ipk}_B)^{\mathsf{ek}_A} || (\mathsf{prepk}_B)^{\mathsf{ek}_A} || (\mathsf{ephpk}_B)^{\mathsf{ek}_A}$$

using newly-generated randomness ek_A.

The master secret yields the first *chain key* $\mathsf{ck}^{1,1}$; the latter will be input to a KDF in order to output a new key $\mathsf{ck}^{2,1}$. For the message key to be generated, a new ratchet key $(\mathsf{rchk}^{1,1})$ is generated with its corresponding public part. Alice also generates a key pair (t_1, T_1), the public part will be used by Bob to initiate its sending chain (chain $y = 2$). Alice then computes a Diffie-Hellman value with $\mathsf{rchk}^{1,1}$ and ipk_B. As for SAID (described later in 3.3.3.2), the protocol uses a persistent authentication mechanism to enforce that the key evolution is due to the matching partner. Basically, Alice signs with SKM_A the keys she used to continue the chain (both the newly generated ratchet key and the key T_1). This signature concatenated with the DH value and the former $\mathsf{ck}^{1,1}$ derive the first message key $\mathsf{mk}^{1,1}$. As for Signal, Alice includes as metadata all information needed for Bob to recompute the chain.

3.3.2.2 The PCS-security of SAMURAI

Key-material. In order to express the PCS-security of SAMURAI, we need first to characterized the key material. In other words, each key must be categorise in one of our three classes: single stage, cross-stage and cross-session key. We give a summary of the key material in Table 3.1.

The keys used in SAMURAI for a single stage only are:

- the message keys $\mathsf{mk}^{x,y}$
- the chain keys $\mathsf{ck}^{x,y}$
- the master secret ms
- the ratchet key $\mathsf{rchk}^{x,y}$
- and also a particular key used only at stage $(1,1)$, namely ephk.

The cross-stage keys, stored throughout the chain until a next vertical move, is t_y since the responder uses this key to initialise its own chain.

Finally, the cross-session keys (for party P) are the one appearing on multiple sessions (say Alice with Bob and Alice with Charlie). For SAMURAI those keys are:

- the identity key ik_P
- the prekey prek_P
- the signature key SKM_P.

Theorem 3 *Consider the* SAMURAI *protocol modelled as a SCEKE scheme. 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 $\hat{S}$:*

- SAMURAI *is* $(1,0)$*-PCS secure against passive attackers, local and medium active outsiders;*
- *For all other adversaries,* SAMURAI *is* (∞,∞)*-PCS secure.*

The results of the theorem are given in Figure 3.6. The proofs are split in two phases, first we show an attack for a given adversary then we proove the further stages are secure.

Passive Adversaries We consider all types of passive adversaries (global, medium, local and insider, outsider). We claim that SAMURAI is $(1,0)$-PCS secure against a passive adversary. Note that this is the best level of security in the sense that only one stage is revealed by the adversary (which is the stage being compromised by definition). Suppose that the strongest adversary (*i.e.*, global passive insider) compromises stage (x,y); thus all keys are revealed but without any other tampering by the adversary (which is passive). However, the next stage of the same chain $(x+1,y)$ is derived through a newly generated ratchet key $\mathsf{rchk}^{x+1,y}$ which is out of reach from the adversary. If we consider the next chain, the adversary can derive the chain keys $\mathsf{ck}^{\cdot,y+1}$ but again, the rachtet keys $\mathsf{rchk}^{\cdot,y+1}$ used to derive the message keys are out of reach. All the security is based on those ratchet keys freshly generated at each stage; the adversary can tamper them by either revealing those keys for each stage or inject its own key material (which is not possible for the passive case).

Local & Medium Active Adversaries Considering those types of adversaries, SAMURAI is $(1,0)$-PCS secure. The adversary, in both cases, has an active capability but no access to the long term keys (*e.g.*, the signature key). The only modification from the previous case

(passive adversary) is that $\mathcal{A}$ can completely intervene in the protocol. So the adversary could inject its own key material in otder to hijack the communication. Yet, the evolution of key is made with peristent authentication meaning that the changes are signed by the current speaker. Since the adversary has no access to the signature key, it won't be able to authenticate malicious key material without being notice by the communicated peers.

Other Adversaries The considered adversaries (active insiders and global active outsider), can *bypass* the PCS and making the protocol unable to heal at any point. This corresponds to a (∞, ∞)-PCS security; the adversary can hijack the communication by injecting its own key material into the communication. For insiders, the attack is the same as for Signal, the adversary is actually the super-user (*i.e.*, the semi-honest server) with active capability. So during registration, the adversary can simply "forget" the keys of a given party and replaces it by its own keys. Thus when a peer wishes to talk to the corrupted party, the adversary is impersonating the receiver completely (and can continue the protocol as usual).

For the global active outsider, the attack is different because the adversary cannot interfere during the registration phase between a party and the super-user (recall that we suppose a secure channel between those two). But, eventually, the adversary can replace the key material since it now has access to the signature key. The attack has the same effect than for insider case.

Security proofs. We now complete our metric results by proving an upper bound of the previous attack (except the ones for which there is no healing). The goal of the following proofs is to show that after (χ, Υ) stages from the reveal query, the communication is back to its security: the adversary can no longer distinguish with non-negligible probability the session key (*i.e.*, the message key) from a random value uniformly distributed.

We assume that all KDF calls are modelled as random oracles. Recall also that our security statements are given by a maximal number of stages $\mathsf{n_S}$, a maximal number of message $\mathsf{n_{x-max}}$ in a given chain, a maximal number of chain $\mathsf{n_{y-max}}$ run by any given instance, a number of parties generated by the adversary $\mathsf{n_P}$ and a number of sessions n_π created by any given party.

There are two cases to consider for the type of adversary:

- passive adversary: this case regroups in fact the 3 passive insider adversaries and also the 3 passive outsider adversaries. The difference between the insider and outsider concerns access or not to the super-user data. When looking at an upper bound, any argument with stronger adversary is also true with weaker adversary. So our proof for the passive insider implies a proof for the outsider case.

- medium active outsider adversary: this case implies a proof for the local adversary since medium reach is stronger than local reach (recall that the set of keys which can be revealed by a local adversary is included in the set of keys for the medium adversary).

Note that those two cases are distinct without any possible implications from one to another. This comes from the fact that in the first case, the adversary is stronger in terms of access (insider has more capability than outsider) while the power is weaker (passive against

active). There is no clear hierarchy a priori [7] in those two types of adversary.

$\mathbb{G}_0$: This game corresponds to the original security game (Figure 3.5 of Sec. 3.2.4). The advantage of $\mathcal{A}$ against this game is Adv_0.

$\mathbb{G}_1$: In this game 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 oTest.

If $\mathcal{A}$ queries another parties, session or tested stage then 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 and signature key for each party.

$\mathbb{G}_2$: This game is the same as $\mathbb{G}_1$ except that the challenger aborts if two values 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 ms is guaranteed. Indeed, ms is computed using ik and also ephk thus by uniqueness of the former and the latter, we have uniqueness of the shared secret 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 rchk. We have:

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

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

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

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

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

For clarity, we keep the notation prek_Q instead of prek_Q^i (signed pre-key of index i).

$\mathbb{G}_6$: In this game, we ensure that there is no collision for honestly-generated T_0 . We can upper-bound the total number of such keys by the number of sessions n_π. In addition, we ensure that the challenger guesses the correct index for T_0 (recall that several cross-stages keys are uploaded to the server) thus we have:

[7] This statement depends, in fact, of the protocol. For Signal, the insider gets no further advantage than outsider because the server only stores public keys (without access to the private parts). Yet, for SAID, the server generates the key pairs for each party thus an insider is stronger than outsider. However, an insider has the capability of impersonating the server so if the adversary is active than it is strictly stronger than an outsider.

$$\mathsf{Adv}_5 \leq \binom{\mathsf{n}_\pi}{2} \cdot 2^{-|\mathtt{T}_\emptyset|} + \mathsf{n}_\pi \cdot \mathsf{Adv}_4$$

$\mathbb{G}_7$: We rule out any collision for honestly-generated keys T_y ($y > 0$). We have:

$$\mathsf{Adv}_6 \leq \binom{\mathsf{n}_\pi \cdot \mathsf{n}_{\mathsf{y-max}} \cdot \mathsf{n}_{\mathsf{x-max}}}{2} \cdot 2^{-|\mathtt{T}|} + \mathsf{Adv}_7$$

$\mathbb{G}_8$: In this game the challenger accepts collision of ik_P and rchk. 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:

$$\mathsf{Adv}_7 \leq \frac{1}{q} + \mathsf{Adv}_8$$

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

Passive Insider. The passive insider adversary corresponds to the passive outsider adversary. Indeed, SAMURAI (and also Signal) includes a semi-trusted server which receives the public keys used to setup a session. So an adversary posing as $\hat{S}$ is equivalent to an outsider adversary for the passive case. All attacks in the insider case is alos possible in the outsider case (and vice-versa). for simplicity, we consider $\mathcal{A}$ as outsider.

$\mathbb{G}_9$: We modify $\mathbb{G}_8$ such that the adversary aborts as soon as $\mathcal{A}$ queries the random oracle on ($\bullet \,||(\mathsf{prepk}_Q)^{\mathsf{rchk}_P^{0,1}}$) where the first part of the input is analogous to ms. The random oracle model implies that $\mathcal{A}$ needs to give the exact inputs to retrieve the correct output. If $\mathcal{A}$ can compute such inputs (*i.e.*, $(\mathsf{prepk}_Q)^{\mathsf{rchk}_P^{0,1}}$) then we can construct an adversary $\mathcal{B}$ winning a GDH challenge. Notice that we apply the same game hop of Signal for the local passive outsider ($\mathbb{G}_7$).

The GDH experiment has input (g^a, g^b) and returns g^{ab} with a DDH oracle access which for input (g^x, g^y, g^z) outputs 1 if $g^{xy} = g^z$.

$\mathcal{B}$ simulates $\mathbb{G}_8$ 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 prek_Q respectively. Since $\mathcal{B}$ has replaced long-term and medium-term keys of two parties (where those keys could be used in other sessions), it must ensure a valid simulation for those (non-tested) stages. In either cases, $\mathcal{B}$ randomly chooses the value $\mathsf{ck}^{1,1}$ 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 ϵ_{GDH} the advantage of $\mathcal{B}$ solving the GDH problem, we have:

$$\mathsf{Adv}_8 \leq \mathsf{Adv}_9 + \epsilon_{GDH}$$

$\mathbb{G}_{10}$: We show that $\mathcal{A}$ can query the random oracle on $DH(\mathsf{T}_0, \mathsf{ik}_A)||DH(\mathsf{T}_1, \mathsf{ik}_B)$ with

negligible probability. We split the analysis of such event for the left and right part of the computation and apply the same technique as the previous game (with GDH experiment). Notice that this should be applied for a number of times equal to the maximum number of chains $\mathsf{n}_{\mathsf{y-max}}$:

$$\mathsf{Adv}_9 \leq 2 \cdot \mathsf{n}_{\mathsf{y-max}} \cdot \left[\binom{\mathsf{n}_\pi}{2} \cdot 2^{-|\mathsf{ik}|} + \epsilon_{GDH} \right] + \mathsf{Adv}_{10}$$

$\mathbb{G}_{11}$: We modify the previous game such that if $\mathcal{A}$ queries the random oracle on $(\mathsf{ck}^{x^\star,y^\star} \| \bullet \| (\mathsf{ipk}_Q)^{\mathsf{rchk}_P^{x^\star,y^\star}})$ with $\bullet$ being the signature on elements needed to derive the session (message) key. We apply the same technique as in $\mathbb{G}_9$ to show that if $\mathcal{A}$ can compute $\mathrm{DH}(\mathsf{ipk}_Q, \mathsf{rchk}_P^{x^\star,y^\star})$ then we can construct adversary $\mathcal{B}$ winning a GDH challenge. Note that $\mathcal{A}$ cannot query oReveal.XStage oracle because of the winning conditions; the tested stage cannot be the same as the compromised one (our metric is $(1,0)$ meaning that the security should be recovered from the previous compromised stage). It should also be noted that we do not give condition on $\mathsf{ck}^{x^\star,y^\star}$ since $\mathcal{A}$ might have revealed it via oReveal.XStage from a previous stage and derived it with calls to the random oracle. Finally, we have:

$$\mathsf{Adv}_{10} \leq \mathsf{Adv}_{11} + \epsilon_{GDH}$$

We conclude the proof by noting that $\mathsf{Adv}_{11} \leq 2^{-|\mathsf{mk}|}$ since the only possibility for $\mathcal{A}$ to recover $\mathsf{mk}^{x^\star,y^\star}$ is to guess directly this value.

Medium Active Outsider. The difference from the previous is that now the adversary has active capability (the oSend and oReceive oracles can be used in the malicious mode). Note that $\mathcal{A}$ cannot impersonate the super-user $\hat{S}$ which is equivalent to a passive insider adversary (the super-user only stores public keys so a passive adversary cannot learn information from it).

The games for this case are the same as the passive case except for the last one. Indeed, the $\mathbb{G}_{11}$ ensures that $\mathcal{A}$ cannot compute $\mathrm{DH}(\mathsf{ipk}_Q, \mathsf{rchk}_P^{x^\star,y^\star})$ but here, the adversary has active capability meaning that it could choose its own key material. So instead of the ratchet key $\mathsf{rchk}_P^{x^\star,y^\star}$, the adversary could generate its own ratchet key $\mathsf{rchk}_{\mathcal{A}}^{x^\star,y^\star}$ thus hijacking the session key without Q noticing it.

Yet, the argument from preventing this attack is the signature of the ratchet key (among other values) sent along the encrypted message.

We show that if $\mathcal{A}$ can generates its own ratchet key for the tested stage (recall that it cannot compute the honest value without non-negligible probability of winning a GDH challenge) then we can construct adversary $\mathcal{B}$ winning a EUF-CMA challenge.

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 rchk (for simplicity we omit the other values) 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:

$$\mathsf{Adv}_{10} \leq \mathsf{Adv}_{11} + \epsilon_{EUF-CMA}$$

We conclude by observing that $\mathsf{Adv}_{11} \leq 2^{-|\mathsf{mk}|}$ since the only possibility for $\mathcal{A}$ to recover $\mathsf{mk}^{x^\star,y^\star}$ is to guess directly this value.

3.3.3 SAID protocol

SAID was introduced in 2019 by Blazy *et al.*, whose main goal was to strengthen the authentication guarantees provided in messaging protocols. Another main difference between Signal and SAID is that the latter was constructed in the identity-based (IB) setting, which not only replaces Signal's credential server by a Key-Distribution Center (KDC), but which also comes with some substantial modifications within the protocol.

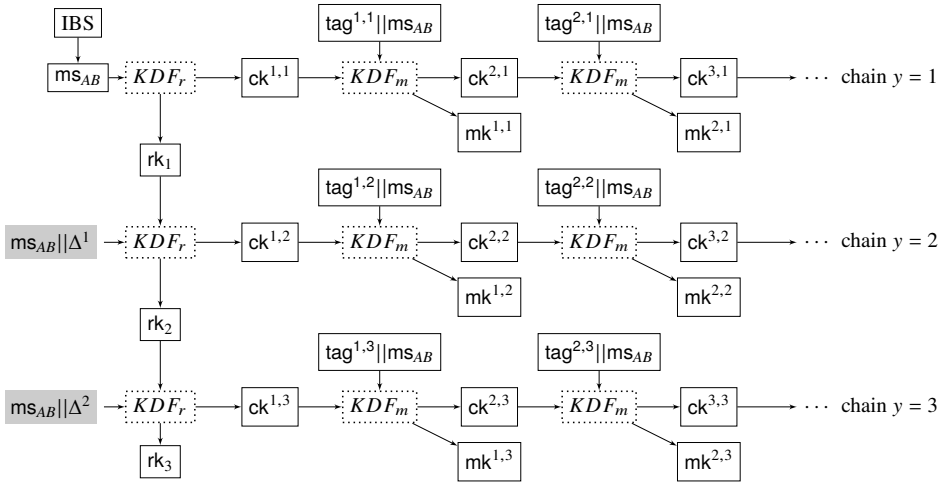


Figure 3.8 *The key schedule of SAID.*

3.3.3.1 The SAID protocol

Unlike Signal, SAID protocol was designed in the identity-based paradigm, allowing parties to derive other parties' public keys from their identities (thus removing the need for certification). In order to associate private keys that are secure and unpredictable to each identity public key, a special entity (the Key Distribution Center) will initially run a global setup, then generate and distribute the private keys to the 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 [FZL10] scheme is made up of four possibly randomized algorithms IBSig=(`aIBS.Setup`, `aIBS.Extract`, `aIBS.Sign`, `aIBS.Vfy`) with the following properties:

- `aIBS.Setup`: takes in input a security parameter λ (in unary) and outputs: public parameters $\mathsf{IBS.ppar}$, a public key $\mathsf{IBS.mpk}$, and a private key $\mathsf{IBS.msk}$;
- `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 σ;
- `aIBS.Vfy:` takes in input the public parameters, the identity P of the purported signer, a message M and a signature σ, 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 to [FZL10] instead.

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 $\mathsf{IBS.ppar}$ and a master key-pair ($\mathsf{IBS.msk}, \mathsf{IBS.mpk}$);
- 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: KDF_r for root- and chain-key derivation, and 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.

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.		
$\mathsf{AD}_{1,1} \leftarrow (\mathsf{meta}_1, h, x = 1, \mathsf{tag}^{1,1}, \sigma)$	$\xrightarrow{c \leftarrow \mathtt{AE}_{\mathsf{mk}^{1,1}}(M_{1,1} \mid \mathsf{AD}=\mathsf{AD}_{1,1})}$	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 $M_{1,1}$.
ℓ-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 $\mathsf{AD}_{\ell,1} \leftarrow (\mathsf{meta}_1, h, (x = \ell), \mathsf{tag}^{\ell,1}, \sigma)$	$\xrightarrow{c \leftarrow \mathtt{AE}_{\mathsf{mk}^{\ell,1}}(M_{\ell,1} \mid \mathsf{AD}=\mathsf{AD}_{\ell,1})}$	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: $\mathsf{meta}_2 \leftarrow (A, B, \mathsf{Rchpk}^2)$ $\mathsf{AD}_{1,2} \leftarrow (\mathsf{meta}_2, (x = 1), \mathsf{tag}^{1,2})$
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 $M_{\ell,1}$	$\xleftarrow{c \leftarrow \mathtt{AE}_{\mathsf{mk}^{1,2}}(M_{1,2} \mid \mathsf{AD}=\mathsf{AD}_{1,2})}$	
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 3.9 *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.*

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 [8] $\mathsf{ID.sk}_A = H_2(A)^{\mathsf{ID.msk}}$. 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 3.9, 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 σ will be part of the associated data (AD) of all the messages sent by Alice along the first 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 rk_1 and chain-key $\mathsf{ck}^{1,1}$, the values (ms_{AB}, g_1) are input to 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 σ, 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 Figure 3.9 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, 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

[8] The user's secret identification key is essentially a Boneh-Franklin key for identity based encryption [BF03].

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 Figure 3.9, assuming A was the sender at stage (x, y), then, to send his response B selects a random ratchet secret key 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 Δ^y, and the current root key (of level $y - 1$) to 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 Figure 3.9, 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 $\mathsf{tag}^{x,y}$.

3.3.3.2 SAID description as SCEKE

Our generic SCEKE protocols feature a more careful modelling of the user-registration process, which is now an interactive protocol, prone to attacks by an adversary. As a result, we add two details to the description of Signal, notably 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 explicitly feature in SAID, Blazy *et al.* [BBB+19] do suppose the existence of a mutually-secure channel during that process.

Setup. SAID requires as a building block an identity-based signature scheme $\mathsf{IBSig} = (\mathtt{aIBS.Setup}, \mathtt{aIBS.Extract}, \mathtt{aIBS.Sign}, \mathtt{aIBS.Vfy})$. It also needs a type-3 pairing e. At system setup the KDC generates some global public and private parameters required for running SAID. In particular, it must generate global setup values ($\mathsf{IBS.msk}$, $\mathsf{IBS.mpk}$) for the IB signature scheme and also 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.

We moreover allow $\hat{S}$ to generate a pair of private/public keys for the key-establishment protocol of its choosing (for instance TLS 1.3), denoting them $\hat{S}.\mathsf{sk}$, $\hat{S}.\mathsf{pk}$, and then appends $\mathsf{ID.msk}$ and $\mathsf{IBS.msk}$ to $\hat{S}.\mathsf{sk}$.

Key generation. In addition to the material described in [BBB+19], we have parties initially

register some non-identity-based keypairs $(\mathsf{ik}, \mathsf{ipk})$, to be used during registration.

Registration. During *registration*, users first establish a secure channel to the KDC (using, on the user side $(\mathsf{ik}, \mathsf{ipk})$ and on the super-user side, $(\hat{S}.\mathsf{sk}, \hat{S}.\mathsf{pk})$). Over this channel, the user (say Alice) sends her identity A (for instance a phone number, an email address, etc.), 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 $\mathsf{ID.sk}_A = H_2(A)^{\mathsf{ID.msk}}$. In other words, the KDC will know all the users' private keys.

Instance initialisation. In SAID, the parties no longer need the super-user to instantiate a session. Its contribution is thus deemed void. The session's initiator Alice will choose some randomness r and compute $\mathsf{ms}_{AB} = e(H(B), \mathsf{ID.mpk})^r$: in other words, Alice embeds the identity of Bob into the master secret. Alice also generates a random *tag*: $\mathsf{tag}^{1,1}$, and uses it together with the master secret to derive the root key rk_1 and the first chain key $\mathsf{ck}^{1,1}$. The presence of a fresh tag at each stage is a specificity of SAID, which will ensure that keys are unlikely to repeat.

As in Signal, a KDF is then used to obtain the second chain key $\mathsf{ck}^{2,1}$ and first message $\mathsf{mk}^{1,1}$ from the input values ms_{AB} and $\mathsf{ck}^{1,1}$.

Unlike in Signal, in SAID the master secret value ms_{AB} is used at every stage, thus requiring the knowledge (on the part of Bob) of his private identity-key, and on the part of Alice, of the secret r that was signed with her IB signing key. In [BBB+19] all parties P store values 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 in our work.

Sending and receiving. Stages and keys evolve in SAID in a similar way as in Signal:

- **Symmetric ratcheting:** To go from stage (x, y) to $(x + 1, y)$, the current speaker generates a new tag $\mathsf{tag}^{x+1,y}$ and then uses it, together with the chain key $\mathsf{ck}^{x+1,y}$, in order to output $\mathsf{ck}^{x+2,y}$ and $\mathsf{mk}^{x+1,y}$. In actual fact, the process contains two substeps, which are detailed in Appendix 3.3.3.1.
- **Asymmetric ratcheting:** When speakers change, the key material is freshened up with Diffie-Hellman randomness (like in Signal). The key schedule for the new chain takes in input the master secret, a value $\Delta^{y+1} = DH(\mathsf{Rchpk}^{0,y-1}, \mathsf{Rchpk}^{0,y})$, and the root key rk_y, outputting rk_{y+1} and $\mathsf{ck}^{1,y+1}$. The chain key, master secret, and a freshly generated tag $\mathsf{tag}^{1,y+1}$ are fed to a KDF in order to obtain the first message key of the chain, $\mathsf{mk}^{1,y+1}$.

As in the case of Signal, the public key material that allows the receiver to ratchet correctly is sent as metadata. For the first chain of messages, Alice will send the following values as Associated Authenticated Data (AAD): the public value $h = g_2^r$ corresponding to the secret r that the initiator fed into the master secret computation; the the stage's horizontal index $x = 1$; metadata consisting of a public ratchet key $\mathsf{Rchpk}^1 = g_1^{\mathsf{rchk}^1}$, the tag of the current message, the user identities; and a signature over all those values 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 $y = 2$, the auxiliary data is of the same form, except that of course we no longer need a value h. Starting from $y = 3$, another auxiliary value is added:

	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 3.10 *Comparison of leakage oracles for SAID (our framework and [BBB+19]). We denote by "BB" a black-box access to an oracle.*

the number N_{y-2} of messages that the sender sent in its previous sending chain (*i.e.*, chain $y-2$). Recall that we depict the key-schedule of SAID in Figure 3.8.

Comparing security models. Our framework follows closely the model by Blazy *et al.* [BBB+19], 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 [BBB+19] 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, [BBB+19] gives different leakage possibilities to its adversaries than we do, through three specific oracles (presented in the lower half of Figure 3.10). 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 oReveal.XStage yields all cross-stage keys together). Finally, [BBB+19] allows $\mathcal{A}$ black-box access to a long-term value: we denote this in Figure 3.10 by a "BB" annotation. We describe in detail our classification of keys as stage-local, cross-stage, and cross-session in Table 3.1 and in the following paragraph.

Although oReveal provides $\mathcal{A}$ more fine-grained access to the local and cross-stage keys (as it can reveal them 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 [BBB+19] by a oCorrupt query) simultaneously with the chain or root key allowing $\mathcal{A}$ to compute a target message key.

3.3.3.3 The PCS-security of SAID

Key-material. We divide the key material input into the key-schedule of SAID in a similar way as we did with Signal. The results, also shown in Table 3.1, are the following:

- **Stage-specific keys:** In this category, we have chain and message keys as in Signal, together with the newly-generated tags. In addition, we feature the randomness r used at the beginning of the protocol, within the computation of the master secret;
- **Cross-stage keys:** Interestingly enough, the master secret is now a cross-stage key,

because it is input at every stage of the protocol, not just the first one. Other cross-stage keys include root and ratchet keys;

- **Cross-session keys:** The long-term keys of the protocol are: the initial private key ik, as well as identity-based signature and identity keys (denoted ID.sk and IBS.sk).

Theorem 4 *Consider the SAID protocol modelled as a 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* 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* (∞,∞)*-PCS secure.*

Analysis. The results regarding SAID in this paper are partly based on the proofs by Blazy *et al.* [BBB+19]; however we note that their security model is slightly different from ours. For instance, Blazy *et al.* use a trusted execution environment to safeguard all the values of ms that a party ever computes in its lifetime – thus essentially placing it at the same level as the party's long-term keys. We choose to keep the master secret at a cross-stage level, where it technically belongs. In so doing we quantify the security of SAID when a TEE is not employed.

Local Outsiders We consider passive and active outsiders together. We claim that SAID is $(1,0)$*-PCS-secure* – which is actually optimal in our framework. The main reason this is the case is that the master secret value is input in SAID at every ratchet; hence, denying the adversary access to it results in denying $\mathcal{A}$ the ability to make keys evolve without further corruption. This guarantees the optimal PCS security of SAID.

Other passive adversaries For global and medium passive insider and outsider adversaries the PCS security limitations are given, as for Signal, by the fact that, on the one hand, the adversary is able to learn a ratchet key $\mathsf{rchk}^{x,y}$ at the last corruption stage (x,y), and 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 $\hat{S}$.sk is not helpful. In this case, SAID is $(\infty,2)$*-PCS secure*.

In the case of a local passive insider, as in the case of Signal, the adversary is crippled by its lack of knowledge or root and ratcheting keys.

Other active outsiders Knowledge of the master secret is fundamental in SAID. Since we do not consider TEEs, both medium and global active outsiders are allowed access to the master secret, and can hijack the session by including fresh asymmetric ratcheting elements once the corruption has been done. In this situation, as in Signal, the protocol never heals ((∞,∞)-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 ((∞, ∞)-PCS security).

Security proofs.

$\mathbb{G}_0$: This game corresponds to the original security game (Figure 3.5 of Sec. 3.2.4). The advantage of $\mathcal{A}$ against this game is Adv_0.

$\mathbb{G}_1$: In this game 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 oTest.

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

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

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 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 ck and mk. First we show that the master secret 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 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 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:

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

$\mathbb{G}_4$: This game is the same as the previous except that the challenger aborts if $\mathcal{A}$ queries

the random oracle with 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 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 $\mathsf{n}_{\mathsf{y-max}}$ 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 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 3.2.4, 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 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 α 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 ms_{PQ} to the random oracle. Thus we have:

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

Finally, if $\mathcal{A}$ never sends 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 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 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 (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 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 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).

3.3.4 Application to 5G-handover

We showcase the flexibility of our SCEKE framework by describing how it can capture a host of secure-channel establishment protocols in 5G, including the famous AKA and a series of procedures called *handovers* [3GP20, 3GP21]. If asynchronous messaging (like Signal and SAID) allows for two communicating partners, these 5G protocol are in fact run between various entities. However, by focusing on the end-users within the secure-channel establishment routines, we can nonetheless fit the 5G secure-channel establishment primitives onto a SCEKE protocol that enables a secure channel be created, and then maintained through key-evolution.

We begin by first describing the 5G context and how handovers are actually run in this environment; then we show how to model these 5G secure-channel establishment procedures as a SCEKE scheme and analyze their PCS security.

3.3.4.1 The 5G handover protocols.

The 5G technology is vast with numerous applications, services, protocols etc... We chose to focus on peculiar procedures to this use-case: the handover procedures. In a nutshell, those procedures occur when some independent components (which will be called gNB) share

the management of a given client (called UE). With the consecutive technical specifications along the years, several handover procedures have been deployed as illustrated in Figure 3.12. While a complete description of all those procedures could be insightful, we rather focus on two of them: the XN and N2 procedures. This choice is first motivated by the fact that describing and analysing all the procedures may be not feasible in this work; a second motivation comes from the practicality of these procedures, most of the handover are done using either XN or N2. Finally, the choice of the 5G handover procedures for our metric (instead of other popular asynchronous messaging protocols like Wire, Matrix or even TLS 1.3) may be surprising. The goal is to present the strengths of our model with protocols that are not analysed toward PCS. While Signal (and SAID) is well-known and well-analysed protocol (allowing us to ascertain the correctness of our model), the handover procedures are not as well popular (at least in the academic field). Thus we wanted to fill the gap between this lack of analysis by the academic and the practical point of view of those procedures.

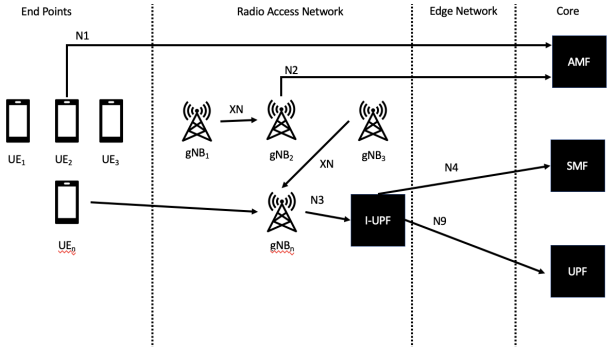


Figure 3.11 *Handover procedures in 5G networks.*

We present an overview of the 5G networks[9] and outline of the AKE procedures that occur therein. We refer to this composition of AKE procedures as the 5G-SCEKE.

When we present it, we focus just on elements relevant to our metric and analysis (i.e., we ignore a series of messages exchanged therein that have no impact here).

An Overview of the 5G Network. The customers in a mobile network are in possession of a mobile phone or device containing a SIM. The device and SIM together are referred to as *UE*. This UE can run different procedures (e.g., radio protocols, cryptographic ones) with other parts of the mobile network. The SIM inside the UE contains identifiers and cryptographic material shared only with a few essential entities/servers of the *core* of the network managed by the *operator* (e.g., Vodafone, Orange).

The *core* is an essential part of the operator's network; we present this as a whole, as its inner servers can be treated holistically for this work.

All access of the UEs' to the service and to the core is done via a network of radio "base-stations" which, in 5G, are referred to as *gNB*. At any point, any UE is served by one gNB.

[9]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.

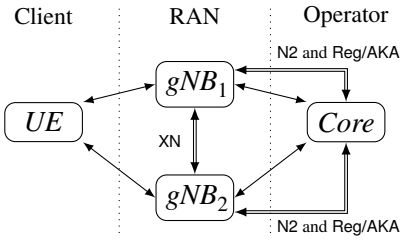


Figure 3.12 *On overview of the main 5G entities & Procedures*

Entities in the 5G network are linked via one or more *interfaces*. All interfaces between the gNBs themselves, as well as between the gNBs and the *core* are considered to be secure channels (i.e., ensuring confidentiality, integrity and authentication); the security of gNB-to-gNB and gNB-to-core interfaces does not make the direct subject of this work.

Securing the 5G Access-stratum Channels. The protocols securing the channel on interface between the UEs and the RAN (i.e., between a given UE and its serving gNB) are of interest herein. A series of AKE protocols, which we collectively call *5G-SCEKE*, are used to re-fresh the *access-stratum (AS) keys* which provide confidentiality (via encryption), authentication and integrity (via MAC-ing) for the channel between any given UE and its serving gNB. In fact, the protocols in the *5G-SCEKE* refresh not just the AS keys, but a series of short-term keys in a hierarchy of 5G keys. Of these refreshed keys – at any given point– some are shared between a UE, one or two gNBs and the *core*, and others are shared just between a UE and the *core*.

The *5G-SCEKE* is formed of: *Registration*, via the AKA procedure, and *handover* procedures – via XNp and N2p, both of which come with variations in the key-derivation mechanisms. Each such procedure is executed between one UE, one or two gNBs and the *core*.

An overview of which entities run which procedures (aside of the UE which runs them all) is given in Figure 3.12.

The *Registration* & AKA Procedures. The Regp procedure is described mainly in [3GP20, 3GP21]. In this subsection, we do not focus on the whole description of the message flow in the Regp procedure, but on aspects of key-establishment done within.

The Regp procedure is executed when a UE (re)gains signal. Then, this UE will be served by a physically close-by gNB which we denote *s-gNB*. The Regp procedure is run between said UE and the core, and it is *passively* proxied by said *s-gNB*.

The $\mathbf{K_{AMF}}$ Key. The Regp procedure is split in three parts, with the second part being the AKA procedure. During this, the core authenticates the UE and the two re-fresh a list of short-term keys with a key hierarchy, some used just as key "seeds" for other keys (e.g., NH) or authentication purposes (e.g., K_{SEAF}), whilst others (e.g., K_{AMF}) being used in actual exchanges between the UE and the *cire*. The key of most interest herein, regenerated by the UE and the core at the end of AKA and being the last key in the *"AKA key-hierarchy"* is called $\mathbf{K_{AMF}}$.

The $\mathbf{K_{gNB}}$ Key. At the end of the Regp procedure, out of K_{AMF}, the core and the

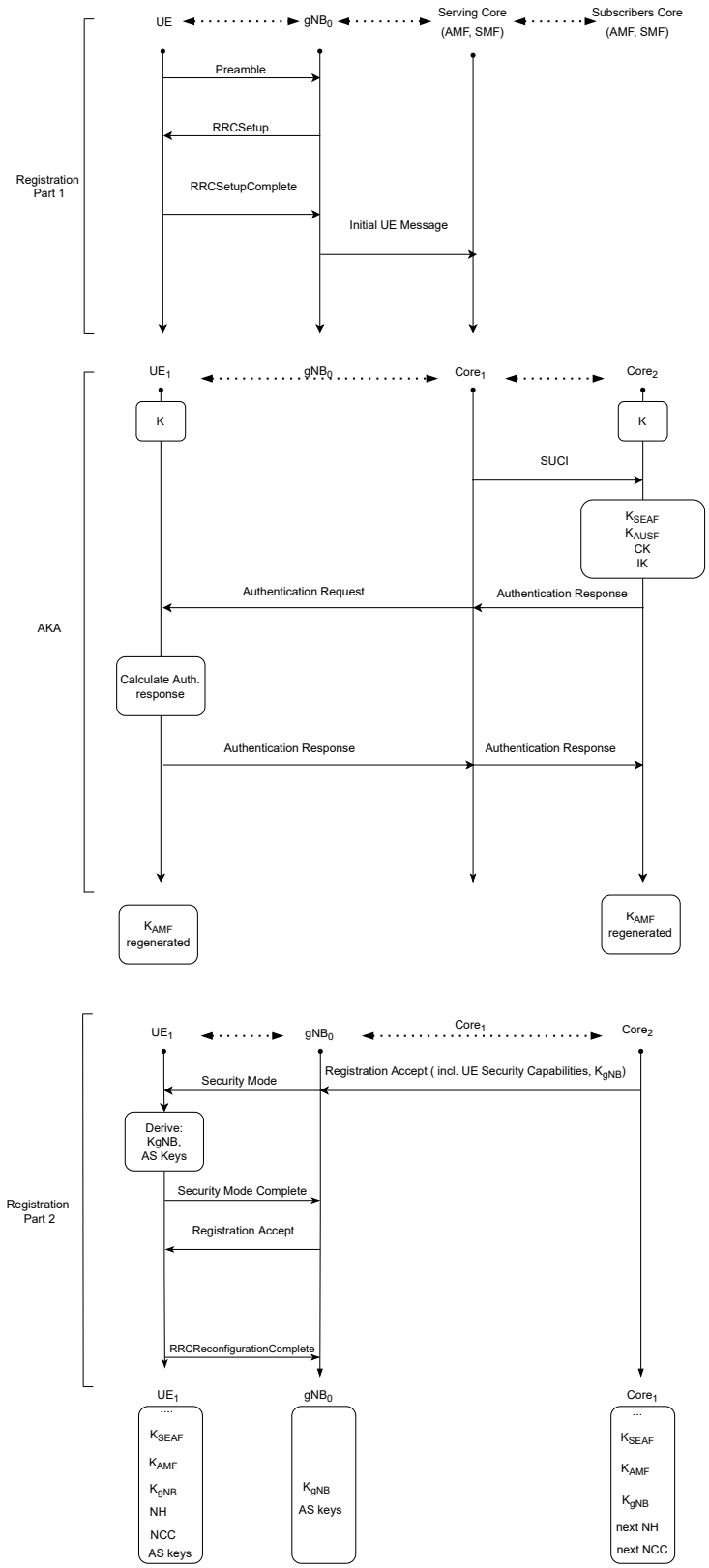


Figure 3.13 *The* 5G-SCEKE *Short-term Keys' Update and Distribution at the End of Regp (. . . denote existence of other shared keys)*

UE generate another key – called the ***security key*** and denoted $\mathbf{K_{gNB}}$. The core sends K_{gNB} to *s-gNB*, and this radio node uses K_{gNB} to derive the access-stratum keys (e.g.,

K_{UPINT}, K_{UPENC}) to communicate securely with the UE henceforth; so does the UE. From here on, the communication between the UE and *s-gNB* is secured via the AS keys derived from K_{gNB}. To denote the status at the end:

An equivalent message to this Regp run is given in Figure 3.13.

The *Handover* Procedures: XNp and N2p. The handover procedures are described mainly in [3GP20, 3GP21].In this subsection, we do not focus on the whole description of the message flow in the handover procedures, but on aspects of key-establishment done within.

A handover procedure is executed every time the UE swaps from being served by one gNB, called *sgNB* to being served by another gNB, called *tgNB*. This generally happens in order for the UE to get better signal/connection, as it physically moves away from one radio node and closer to another.

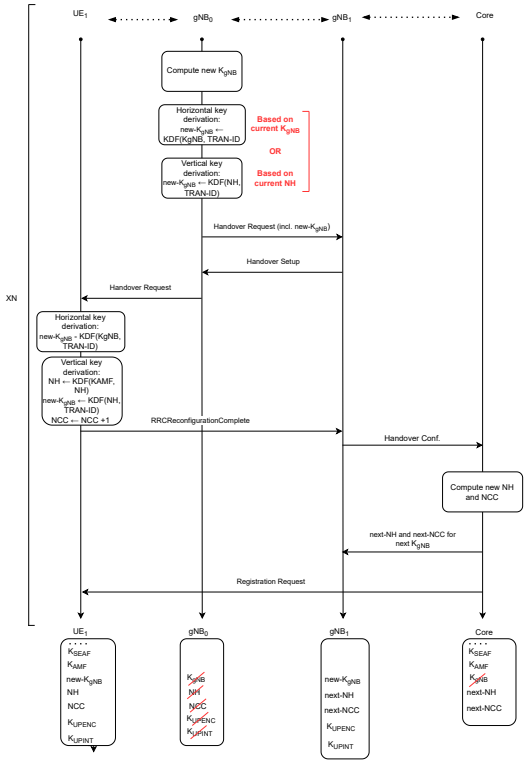


Figure 3.14 *The* 5G-SCEKE *Short-term Keys' Update and Distribution at the End of XNp (Red cross denote lack/deletion of keys; . . . denote existence of other shared keys)*

Two Types of Handovers. To communicate securely, new AS keys need to be established between the UE and the target node tgNB. For this, as explained for the Regp procedure, a new security-key K^1_{gNB} needs to arrive on the target node tgNB. 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 tgNB, similarly to the the Regp procedure; in this case, the source node is a passive proxy. This procedure is the N2p protocol.

2. The source node sgNB, 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 tgNB a new security-key K^1_{gNB}. This procedure is the XNp protocol.

"Horizontal vs. Vertical" XNp Protocol & Backward Security. Due to the key-derivation used to yield the AS keys out of the K_{gNB} keys, in the XNp 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 tgNB; this is a well-known fact, and we say that XNp 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})***.

Indeed, consider a series of n executions of XN$^{\text{hkd}}$. Note that a new security key K^n_{gNB} for the target node depends on the K^{n-1}_{gNB} key of the source node and other recoverable/public data. So, a rogue gNB who was source node n handovers ago can retrieve iteratively all the security keys $K^1_{gNB}, \ldots, K^n_{gNB}$ and therefore decrypt the UE access-stratum traffic with all the nodes that were target nodes in these n handovers. So, XN$^{\text{hkd}}$ executed in series has a systematic loss of backward security.

However, there is an alternative. At the end of each handover, the core sends to the target node a fresh key called NH. This is derived by the core from K_{AMF}, which –as we explained– is refreshed (at least) at the end of each Regp execution. When a gNB is source node in an XNp 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[10]. This type of key-derivation inside XNp, 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})***.

The XNp's key-derivations are also given in Figure 3.14.

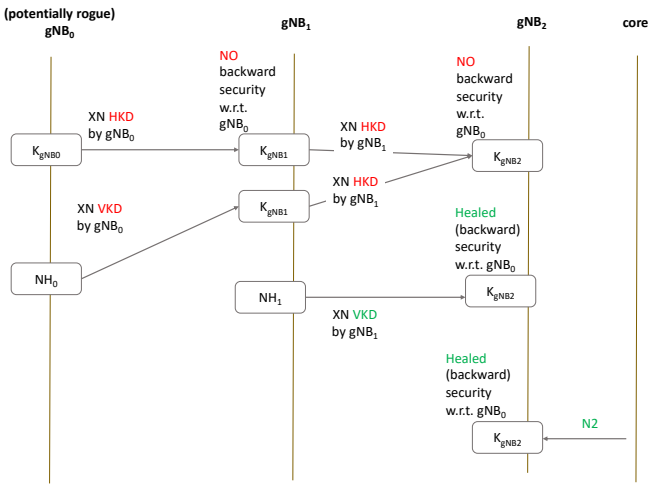


Figure 3.15 *Overview of Backward Security w.r.t. K^*_{gNB}: Loss and Healing via 5G Handovers*

[10]The NH could have been used by the current sgNB, if this gNB received it at the end of N2p, or if the served UE had temporarily been in IDLE state.

Clearly, in a sequence of XNp executions, as soon as a vertical key derivation takes place, the chain of serial loss of backward security is broken. Please refer to Figure 3.15 for a visual representation of this backward security lack and post-compromise healing'.

The N2p Protocol & Better Backward Security. The N2p protocol requires less trust in the gNBs, since the source node is not actively calculating the security-key for the target node. In N2p, 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 tgNB; this preservation of backward security of K^*_{gNB} by N2p is also represented in Figure 3.15.

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 *" N2p with K_{AMF} rekeying"; we refer to this as* $\mathsf{N2_r}$. *When we write simply N2p, we generally assume implicitly it is without K_{AMF} rekeying. We specifically refer to* " N2p without K_{AMF} rekeying" *as* $\mathsf{N2_\emptyset}$*).*

The N2p's key-derivations are also given in Figure 3.16.

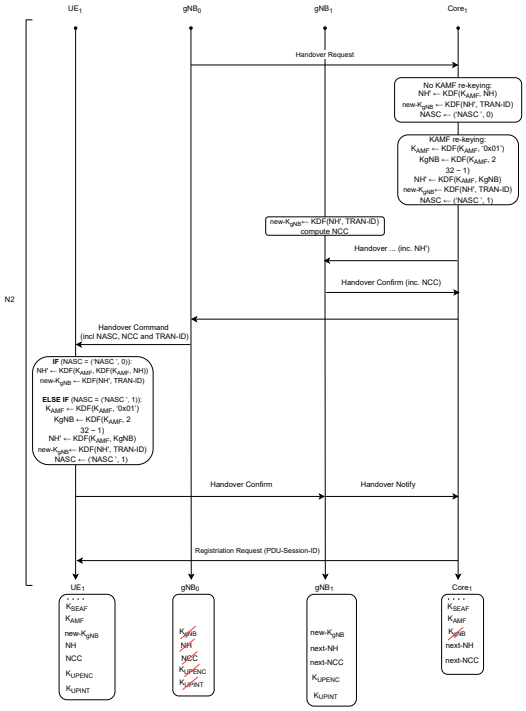


Figure 3.16 *The* 5G-SCEKE *Short-term Keys' Update and Distribution at the End of N2p (Red cross denote lack/deletion of keys; . . . denote existence of other shared keys)*

The choice between the one type of handover or another, i.e., XNp vs N2p, is down to the interfaces between sgNB and tgNB: if there is an XN interface between them, then XNp is executed; otherwise, N2p will be executed, intermediated by the *core*.

Figure 3.17 shows an overview of which procedures in *5G-SCEKE* refresh which keys in the 5G hierarchy.

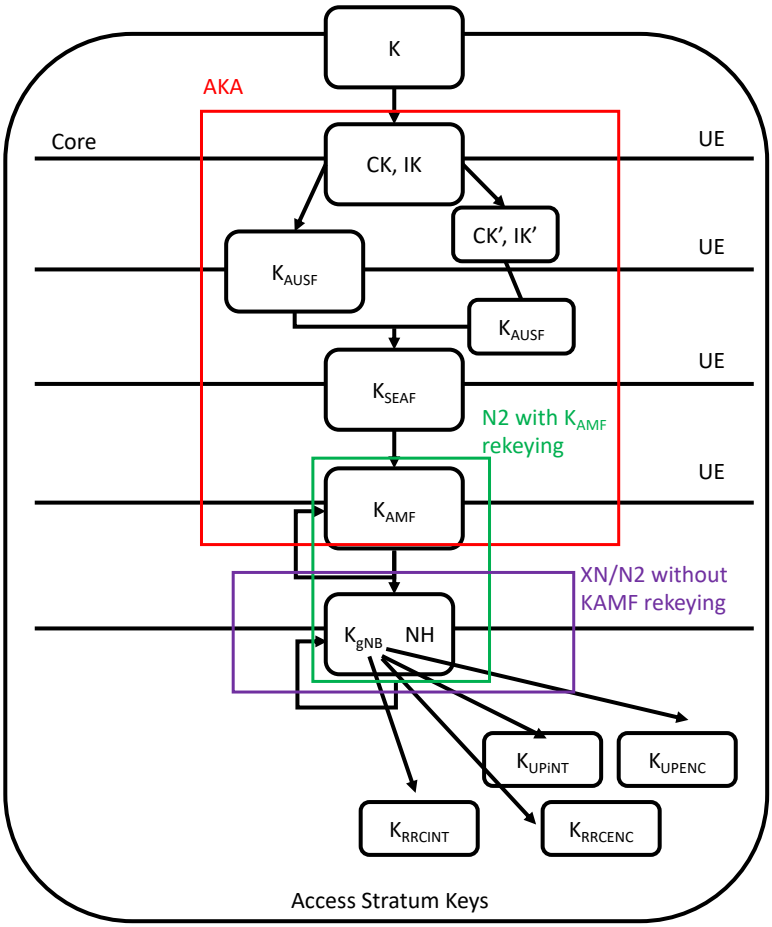


Figure 3.17 *Key Hierarchy and Refreshing in* 5G-SCEKE

3.3.4.2 The *5G-SCEKE* protocol.

We refer to the composition of an initial registration procedure (through AKA) and subsequent key-evolutions (through handovers) as *5G-SCEKE*, and proceed to model it as a SCEKE protocol.

Our SCEKE framework only supports two-party protocols. Thus, we *compress* the set of all gNB nodes and the core network into a single entity (which will represent the responder, Bob[11]). The initiator of the protocol is Alice (the *UE*). The super-user is a key-escrow entity, which basically associates initial key-material to sessions (abstracting the AKA procedure).

Figure 3.18 presents *5G-SCEKE* from the viewpoint of our framework's stages and as SCEKE protocol. Following the initial registration phase, Alice (the initiator) can horizontally evolve keys by using the XN procedure. When Bob wishes to respond, it runs the procedure N2 to evolve the stage vertically. Unlike the somewhat symmetric roles Alice and Bob play in key-evolution for Signal or SAID, in the *5G-SCEKE* protocol the roles are decidedly asymmetric: only Alice evolves stage-keys horizontally, and only Bob evolves

[11] We can do this at no risk within our framework because in the PCS game the Bob entity must be honest for the target session – even if it might have had its secrets revealed. This abstraction, however, is not without loss of generality for notions in which parts of Bob might be malicious.

them vertically.

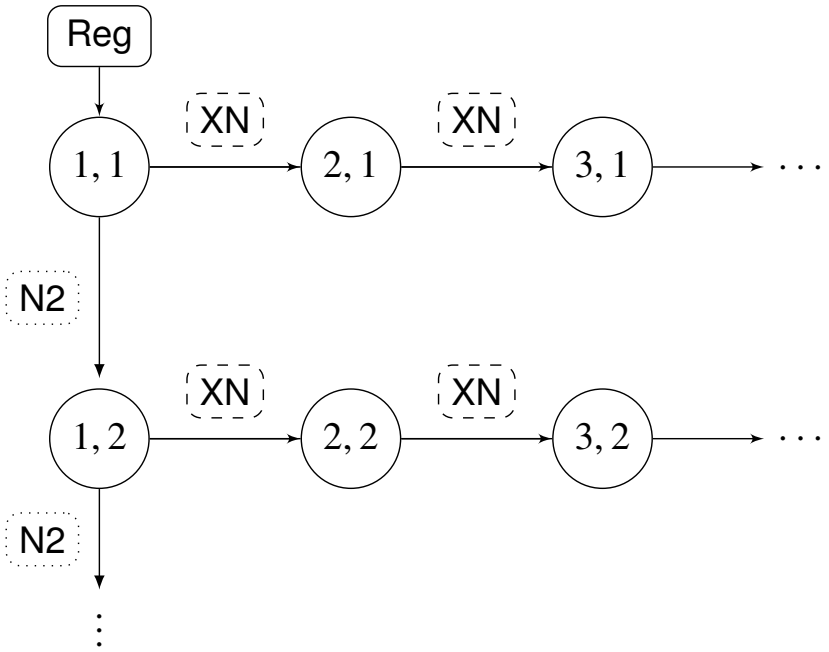


Figure 3.18 *General model of 5G handover procedures.*

5G-SCEKE instantiates the initial steps of SCEKE with:

- **Setup:** Our 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 also 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 channel with $\hat{S}$ and sends a registration request. The super-user generates one secret $\mathsf{K}PQ$ for each Q in its database, but does not yet 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.

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 newly-established secure channel – this key acts as a master secret. The master secret is fed through a KDF to obtain a root key (in practice, $\mathsf{K}^{)}_{AMF}$ and, through an intermediate KDF derivation of a key $\mathsf{NH}^{0,1}$, the first *chain key* $\mathsf{ck}^{1,1}$ (in practice K_{gNB}) and a new root key rk_2 are computed. The latter yields a *message key* $\mathsf{mk}^{1,1}$ (in practice called K_{AS} in the *5G-SCEKE* key-schedule) and $\mathsf{ck}^{2,1}$. Messages sent from Alice to Bob will be encrypted and authenticated with the message keys.

Upon receiving one of Alice's messages, which carries her identifier as metadata, Bob will initialise a session in the same way as Alice.

Sending and receiving messages. Messages are sent and received in stages, encrypted with key material that evolves:

- **Horizontal evolution:** this occurs when the session's initiator wants to send a new

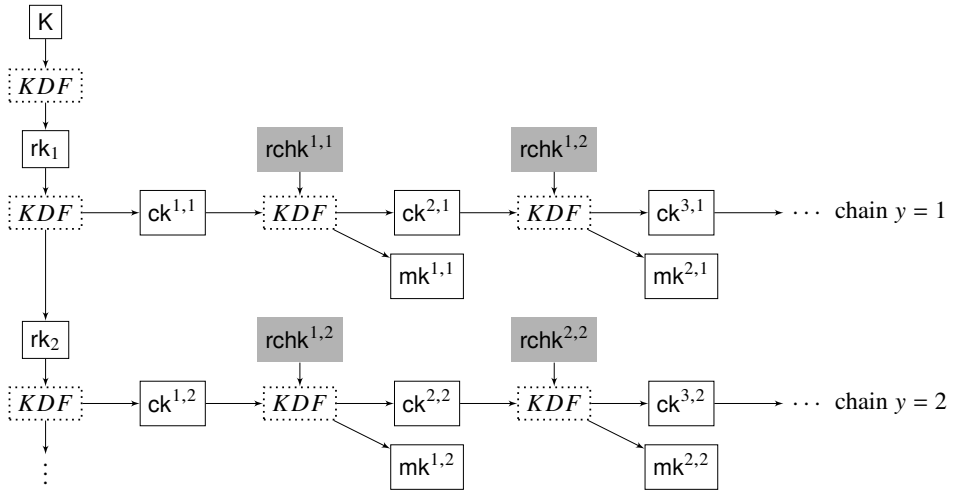


Figure 3.19 *Generic key management for* 5G-SCEKE. *The values in grey are modifications only for* 5G-SCEKE$^+$.

message (the stage evolves from (x, y) to $(x+1, y)$). As in Signal and SAID, the chain key $\mathsf{ck}^{x+1,y}$ (already derived) will be fed into a KDF to obtain $\mathsf{ck}^{x+2,y}$ and $\mathsf{mk}^{x+1,y}$.

- **Vertical evolution:** when the session's responder sends a message (the stage evolves from (x, y) to $(1, y+1)$), a vertical key derivation occurs. A new $\mathsf{NH}^{0,2}$ is generated from rk_2 and both keys are fed into a KDF in order to derive $\mathsf{ck}^{1,y+1}$.

Note that the SCEKE protocol we chose to exhibit based on *5G-SCEKE* and using K_{AMF} is not unique in the 5G context. However, we could apply our framework to multiple variations of such protocols too: for instance using N2 for horizontal evolutions, and Reg (or AKA) for vertical evolutions. In this case, we can discuss healing even in the case corruptions/reveals keys above K_{AMF} in the 5G key-hierarchy.

Our Improved-PCS *5G-SCEKE*. We propose a simple, yet effective, modification of the *5G-SCEKE* to enhance the PCS-security. Noting that in *5G-SCEKE* the lack of (local) at key-evolution allows an adversary to basically ratchet horizontally through the entire chain, we propose to add freshness into this derivation process, thus limiting the attacker's power..

Concretely, we change XNp into what we call ***XN***$^+$. In XN$^+$, sgNB does not simply calculate the k_{gNB} key for tgNB. Instead, the latter contributes to the k_{gNB} with a locally generated secret v. Then, tgNB sends the value v to the core who sends it to the UE, encrypted with k_{AMF}. So, the UE can now compute the new k_{gNB} just as tgNB can. Note that the sending of v can be done on existing XNp messages, so the protocol is minimally modified. We refer to the *5G-SCEKE* composed with XN$^+$instead of XNp as 5G-SCEKE$^+$.

3.3.4.3 PCS-security of *5G-SCEKE*.

We divide the key material input into the key-schedule of *5G-SCEKE* viewed as a SCEKE protocol:

- **Stage-specific keys:** In this category, we have chain keys – corresponding to K'_{gNB} and message keys – corresponding to K_{AS}.
- **Cross-stage keys:** The root key – corresponding to K_{AMF}, and NH key are both

computed at the beginning of each chain and stored for next vertical evolution.

- **Cross-session keys:** The long-term key is the pre-computed shared key K between the two parties. Each registration procedure corresponds to new instances of all the aforesaid keys, where K is used again.

These keys' classification is recounted on Table 3.1.

Theorem 5 *: 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* (∞, ∞)*-PCS secure against insiders, medium and global outsiders.*

We now explain the results given in Figure 3.6 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 (∞, ∞). For this attacker, the reveal of one root key 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 rk_y, or a fortiori from the long-term key K.

Other Insider Adversaries The protocol in this case is (∞, ∞)-PCS-secure. For this attacker, no healing is possible since the attacker has access to keys "above" K_{AMF} in the 5G hierarchy, i.e., above 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 rk is not revealed from a call to $\mathsf{oReveal.1Stage}$. 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.

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.

PCS-security of *5G-SCEKE*$^+$. The key-material is the same as for *5G-SCEKE* except that a new key is added, namely the single-session key $\mathsf{rchk} = v$.

Theorem 6 *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* (∞, ∞)*-PCS secure.*

Again, we 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 K_{AMF} (*i.e.*,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 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 (∞, ∞)-PCS-security.

Insider Adversaries Here no healing is possible, the protocol is (∞, ∞)-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.

Security proofs.

$\mathbb{G}_0$: This game corresponds to the original security game (Figure 3.5 of Sec. 3.2.4). The advantage of $\mathcal{A}$ against this game is Adv_0.

$\mathbb{G}_1$: In this game, the challenger 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 oTest.

If $\mathcal{A}$ queries another parties, session or tested stage then 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 rchk (i.e., two nonces $\mathsf{rchk}^{x,y}$ used in $\mathtt{XN}^+$), used inside the derivation of cks (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 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 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:

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

$\mathbb{G}_5$: 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 ck (i.e., the k_{gNB}s) as the above are unique until the randomness repeats itself (as the KDF producing 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 ck^is considered an RO (as well as from the uniqueness of rchk values in $\mathbb{G}_2$, and from the randomness of rk values in $\mathbb{G}_3$). So, we have:

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

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 ck or mk.

Next, the argument is based, in part, on the fact that in our framework any proven bound strictly lower (χ, Υ), 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 ck (i.e., k_{gNB}) on a stage with $y = y^\star$, but to compute a new ck and mk needs the following:

- the next 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 rk.

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

- The next 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 ck and/or mks, after the crucial "reveal"s have been called, and the probability of such guessing is negligible.

3.4 Discussion and Conclusion

This chapter 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, our protocol SAMURAI, 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 Figure 3.6) indicate that optimal security (*i.e.*, (1,0)-PCS security) is achieved by SAMURAI and 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 SAMURAI and 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.

Chapter 4

Deep-Attestation

In a world interconnected at an unprecedented scale, there exists an increasing need for flexible and dynamic networks. In environments such as cloud and 5G networks, virtualization is a cornerstone to easily scale up with user demand. Indeed, virtual machines or virtualized network functions can be instantiated, scaled and migrated on top of any available hardware infrastructure enabling, for instance, better performance or enforced security. However, virtualization arises new challenges because of vertical stringent requirements. For instance in an e-health service strict resources geolocation and resources access control are a must. This is where the process of *attestation* comes in. We propose the first model of an attestation variant called Deep Attestation, along with a solution to find a better balance between security and performances.

Contents

4.1 Introduction

Network Function Virtualization (NFV) is a technology that promises to provide better versatility and efficiency in large-scale networks. The core idea is to move from architectures in which physical machines are set up to perform various roles in a network, to a design in virtual configuration. As such, a machine could be configured and re-configured at distance, and, by judicious use of virtual machines, it could perform a variety of roles within the network infrastructure.

Virtualized platforms are set up in layers, including the following basic components: physical resources, the virtualization layer and infrastructures, *virtualized network functions* (VNFs), and the NFV management and orchestration module. At the bottom of the infrastructure are real, physical components, meant for computations, storage, and physical network functions. The virtualization layer (also called *hypervisor*) manages the mapping between those physical components and virtual equivalents. As such, the NFVs – hosted by virtual machines running inside the NFV infrastructure– never have direct access to the physical resources. Instead, the VNFs access the virtual resources. The NFV management and orchestration module runs the combined infrastructure, including: the lifecycle of the instantiated VNFs, resource allocation for VNFs, or overall management in view of particular, given network services.

4.1.1 Deep Attestation (DA)

Virtualization enables efficient, versatile remote network configuration and administration; however, the fact that multiple virtual processes share resources can introduce hazards to security. One way to ensure that a component runs correctly is by using *attestation*. Attestation is a process complementary to authentication: whereas the latter allows a platform to prove that it is the entity it claims to be, the former ensures that the platform runs a trustworthy code, *i.e.*,it has not been breached. As described in [ETS17], "*Attestation is the process through which a remote challenger can retrieve verifiable information regarding a platform's integrity state.*". An other definition, appearing in [CGL+11], gives the following terminology: " *Attestation is the activity of making a claim to an appraiser about the properties of a target by supplying evidence which supports that claim. An attester is a party performing this activity. An appraiser's decision-making process based on attested information is appraisal*". A property can be for instance software integrity, geolocalisation, access control, etc.

Attestation relies on a *root of trust* (RoT), usually instantiated through a *trusted platform module* (TPM) – or an equivalent mechanism. The root of trust is responsible, amongst other things, for protecting sensitive cryptographic materials (such as private keys) and for running cryptographic operations in an isolated way. The virtualization layer (hypervisor) has direct access to the RoT, but the virtual machines it manages do not; instead they will have access to the RoT by means of *virtual Roots of Trust* (vRoTs). Virtual Roots of Trust are a combination of resources, some provided by the physical RoT, and other managed by the hypervisor, which directs and mediates access to the RoT.

In a nutshell, attestation is a process which allows an independent, remote verifier to

check that a target platform still behaves in the desired way. This is done by first authenticating the RoT, then by comparing a measurement of the current state of the component to a presumably-correct state, as indicated in a *Root of Trust for Storage* (RTS). In addition, a guarantee must be given of the correctness of the RTS, which is done by means of a *Root of Trust for Reporting* (RTR). Functionalities of RTS and RTR can be provided by a TPM. A TPM is an example of implementation that could provide RTR and RTS by leveraging the specific tampering detection properties of its Platform Configuration Registers (PCR) and issuing signed reports, or *quotes*, of their content.

We consider the attestation of two types of components: virtual machines (VMs), such as VNFs, and the hypervisor managing them, whose underlying physical component includes a RoT providing an RTR and an RTS. This architecture is depicted in Figure 4.1.

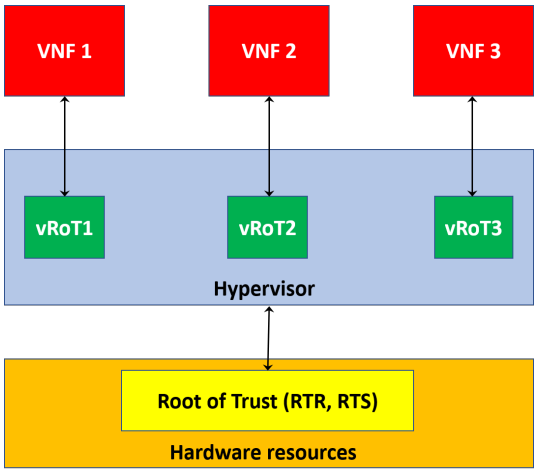


Figure 4.1 *The setup for Deep Attestation.*

To verify that the VMs and the hypervisor are running correctly, both these types of components must undergo remote attestation. First, each component must attest in isolation; then we must attest the *layer-binding* between VMs running on the same hypervisor. This is known as *deep attestation* (DA). There are two typical ways of achieving deep attestation (as described by ETSI standardization documents [ETS17]): single- and multi-channel VM-Based Deep Attestation.

4.1.2 Single/Multi-channel Deep Attestation

In single-channel deep attestation the attestation is run only between the remote verifier and the virtual machines. At each attestation, the VM (by querying its associated virtual TPM, or vTPM) provides not only an attestation for itself, but also the hypervisor it runs on.

Specifically the response forwarded by the VM to the remote verifier includes the (independent) attestation of the hypervisor, and the layer-binding attestation between the VM and its hypervisor. This is depicted in Figure 4.2, on the left-hand-side. Note that the quotes in this case are both obtained from the (slow) physical TPM. From the point of view of security, this solution is optimal; however, it scales poorly. Given as few as 1000 VMs running on top of the hypervisor, we would require that the hypervisor be attested 1000 times, once for each VM.

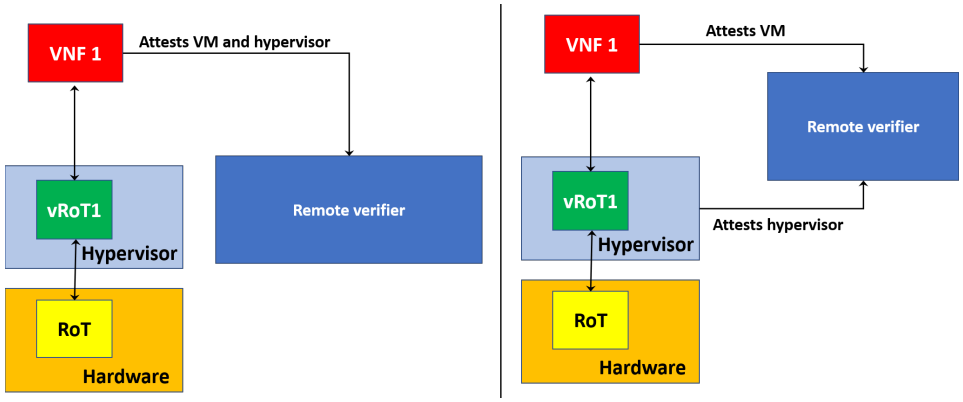


Figure 4.2 *Single vs multi-channel DA*

By contrast, in multi-channel deep attestation, the VMs are attested separately and independently from the hypervisor. In this scenario, the VMs attest to the remote verifier, thus proving they were not tampered with. Separately, the hypervisor also attests to the remote verifier. This can be seen on the right hand side of Figure 4.2. In this case, the efficiency is optimal: for 1000 VMs, we have 1000 VM-attestations and 1 hypervisor attestation. However, there is virtually no layer-binding between the VMs and their hypervisor: there is no guarantee that the VMs are really managed by the hypervisor. An attacker could therefore "convince" a party (such as the owner of the infrastructure) that a VM still exists on a given physical machine when it has, in fact, been removed.

4.1.3 Our solution

We take the middle path between single- and multi-channel deep attestation to obtain layer-binding between VMs and hypervisors with reasonable efficiency. Our solution is simple, yet elegant, using standard cryptography to ensure that a hypervisor's single attestation is linkable to any number of attestations of VMs managed by it. We give three contributions:

A cryptographic scheme. Our scheme ensures secure and efficient linked DA. The hypervisor and VMs each attest only once. However, we also embed a list of public keys (associated with the VMs managed by the hypervisor) *within* the hypervisor attestation, which is established by the root of trust. In order to authenticate the list of forwarded keys, we embed them into the attestation nonce, forwarded by the attestation server. If the hypervisor's attestation verifies, then the attestation server can link that hypervisor with the (subsequently attesting) VMs which use keys in the forwarded list. If the hypervisor's attestation fails, then the public keys cannot be trusted.

Provably secure authorized linked attestation. An important advantage of our approach is that we have a fully-formalized provable-security guarantee. We use a composition-based approach, constructing primitives that are increasingly stronger out of weaker ones. Our goal is to ultimately obtain *authorized linked attestation* (ALA): a primitive which allows components to individually attest (to an authorized entity), and to have their attestations linked. This primitive solves the problem outlined in the introduction, since VMs sharing the same hypervisor will attest in isolation and together with their hypervisor.

ALA schemes will have three properties: *authorization* (only an authorized server can query an attestation quote); *indistinguishability* (no Person-in-the-Middle adversary can know even a bit of a quote exchanged during a legitimate protocol with probability significantly better than $\frac{1}{2}$); and *linkability* (an attestation server can detect if two components are not linked)

We choose to formalize Authenticated Key Agreement (AKA) security as the last of a sequence of primitives, each potentially of independent interest and providing gradually stronger properties. This approach has two virtues: first, we are able to use weaker primitives as black-box components in stronger primitives; and second, the individual proof steps are shorter and smoother.

At the basis of our construction is a yea-or-nay *basic attestation scheme*, which is "secure" by assumption. Its functionality is simple: the basic attestation scheme outputs a faulty attestation whenever a component is compromised, and a correct one for honest components. In other words, this basic attestation scheme is a compromise-oracle: when queried it (indirectly) produces a proof of whether a component has been tampered with or not.

Based on this assumption, we build a sequence of cryptographic mechanisms that adds security against stronger adversaries. A first step is to build *authenticated attestation*: a scheme which allows us to authenticate the component that provides the attestation, and additionally ensures that this component's attestations always verifies prior to corruption, but fails to verify as soon as a compromise occurs. We can think of authorized attestation as the minimum provided (and required) by multi-channel attestation. Then, we consider *linked attestation*: a scheme that introduces the hypervisor-VM relationship described above, and permits not only the verification of individual attestations, but also (publicly) linking attestations.

Implementation. We used a regular laptop equipped with TPM 2.0 (as a root of trust). We set up an architecture with one hypervisor and multiple VMs. The VMs used full virtual TPM as a virtual root of trust. We made over 100 experiments. This showed that our solution is more efficient that single channel approach and adds only insignificant load (a hash function computation) compared to traditional multi-channel DA.

Our work is, to our best knowledge, the first that attempts to provide a sound cryptographic treatment of deep attestation. In many ways, this is much harder than designing the scheme that we present, because attestation is a generic term comprising an entire class of algorithms that have different goals. As such, we are only scratching the surface here, and believe that –aside from the real, and practical advantages of our presented construction– our cryptographic treatment, primitives, and proofs, may be of independent interest to this line of research.

4.1.4 Limitations

A first fundamental limitation is the fact that we assume, in our constructions, the existence of a basic attestation primitive that works infallibly like an oracle, telling us if a component is compromised or not. In reality, this primitive is based on the Platform Configuration

Registers (PCRs) of a TPM. A PCR can store hash digests into a register of the length of the hash function output. Typically a TPM will have multiple banks corresponding to various hash functions (*e.g.*, a sha1 bank and a sha256 bank) with 24 registers for each bank. PCR are reset at each boot and are only updateable through an extension operation $PCR_1 \leftarrow H(PCR_1 \mid H(\mathsf{measurement}))$. We assume the attacker has no physical access to the component and thus cannot tamper with TPM measurements by using hardware attacks. In practice, this is somewhat limiting since we do not account for runtime corruption; thus, the primitive is vulnerable to Time of Check Time of Use (TOCTOU) attacks. Several proposed mechanisms were introduced to monitor runtime integrity, *e.g.*, LKIM [LWPM07] or DynIMA [DSW09]; moreover, in recent years several advancements were made towards verifying runtime integrity for IoT devices [KFZ+20, HHWS18]. Yet, these solutions are not as widely spread at the present day as TPM-based attestation at startup.

We treat the existence of basic attestation as an assumption because we do not see a way of constructing it with cryptographic tools. The cryptography we put on top adds a lot of new properties: authenticity, confidentiality, authorization, linkability, but *not* the simple fact of distinguishing a compromised component from an honest one. Our result should therefore be interpreted as a need for such a scheme to exist, as in fact required by ETSI [ETS17].

Another limitation of our scheme lies in our model of linked-attestation component. We consider classes of components which can be linked. At registration of each piece of hardware, a number of subcomponents of each type is indicated – and (unique) keys are given to those components. As a result, we cannot account for having two hypervisors that manage the same VM on a given infrastructure. A future work could be to consider *multi-hypervisor VM* as introduced in [GKB+19].

4.1.5 Related Work

Many attacks have been recently reported on remote attestation mechanism [BWS19] or 5G standards [HEK+19]. Many tools such as formal methods or cryptography can be used to model and prove the security of such standards. However, this lack of formalization must be now addressed otherwise we will have more and more attacks. Provable cryptography is a nice solution to solve this problem since it allows to better understand the security model, what is the adversary goal and its means, which oracle can he query. Some cryptographic primitives have already be nicely formalized such as Direct Anonymous Attestation (DAA) which enables remote authentication of a trusted computer (TPM for instance) while preserving the privacy of the platform's user in [BCC04] by Brickell et al. It is a group signature without the feature that a signature can be opened, *i.e.*,the anonymity is not revocable. Such primitive are well described using cryptography as a variant of signature scheme. However, provable cryptography has also been used successfully to formalize security protocols as authenticated key exchange [BR93b, CK02]. This is precisely our goal to model the different security components independently and to compose them to prove the security of a new security mechanism. Indeed, the attestation server must authenticate the whole platform, *i.e.*,the hypervisor and the NFV running on top. This problem has been addressed by others in the context of secure boot or for instance in [ABI+15], where the authors propose an

attestation mechanism for swarms of device softwares in IoT and embedded environment. Software attestation is different from remote attestation, as said in [ASSW13] since it cannot rely on cryptographic secrets to authenticate the prover device. The first to have taken into account deep attestation are Lauer and Kuntze in [LK16] but their solution misses a security proof and a rigorous analysis.

4.1.6 Towards authorized linked attestation

Our core contribution provides layer-binding in deep attestation. Cryptographically, we view this as a new primitive, which we call *authorized linked attestation*, built in steps from increasingly-stronger primitives. Each of these intermediate steps plays a double role: on the one hand, it formalizes security guarantees that are of independent interest for attestation (if, for instance, layer-linking is not required); on the other hand, it provides an intuition of the guarantees which specific cryptographic primitives can help achieve.

The first, and basic-most step in our architecture is *basic attestation*, described in Section 4.2. This primitive is an abstraction of the algorithm by which a single party (like a component of a virtualized platform) generates an attestation of its state, given a fresh, honestly-generated nonce. Importantly, basic attestation does not employ cryptography to achieve this feature, but rather, the attestation of registers at startup, using a RoT[1].

Authenticated attestation (in Section 4.3 builds on basic attestation by associating parties with identities. The attestation must now no longer indicate whether the party is compromised: it must also authenticate the component. Here, thus, we enhanced basic attestation with a cryptographic component, which is in fact sufficient to guarantee the basic functionality required by multi-channel attestation. One step further, the linked-attestation primitive (see Section 4.4) built from authenticated attestation will allow two different components to (a) attest their own states; (b) provide auxiliary material that will make two separate attestations linkable. While this primitive has no immediate parallel in real-world attestation, we use it as a handy way of dividing the security proof of our ultimate result into two: linked-attestation will focus on proving the fact that two attestations can be securely linked; whereas authorized linked attestation models attestation as a protocol, using fresh randomness and a secure channel using an honest attestation server.

We also add a new party into the system: the attestation server that serves as a verifier. We then *compose* the linked-attestation primitive with a unilaterally-authenticated authenticated key-exchange protocol, which will authenticate the attestation server and permit the attestation itself to remain confidential with respect to a Person-in-the-Middle (PitM) adversary. All the details of authorized linked attestation is given in Section 4.5.

Finally, we assess the practicality of our solution through a proof-of-concept implementation in Section 4.6. The goal is to compare, in terms of performance, our solution with existing attestation procedures.

[1]To ease notation, we assume that all the registers are attested, and that the property we are attesting is that the entire component has not been compromised.

4.2 Basic Attestation

During basic attestation a single honest party is generated. This party can be later compromised. A quote-generation algorithm will output a quote if the party is still honest at that time, or a special symbol if it is not. Finally a (public) verification algorithm will yield 1 (the component is honest) or 0 (otherwise).

Note that a party such as the one we describe could correspond in practice to a combination of two parties: a virtual entity (like a VM or the hypervisor) and an underlying, incorruptible, secure part (the TPM), which actually generates the quote. At this stage, we importantly do not associate these entities with keys as authentication will only appear in our next step (Section 4.3).

What we want to capture, formalized by the security of basic attestation, is the minimal assumption that a compromised component will always yield an attestation that will fail the verification. This is why, when basic attestation is run for a compromised component, it will yield the special symbol $\aleph$. We also demand correctness: when a non-$\aleph$ quote is generated, the latter will automatically verify. Our basic attestation component thus becomes the minimal non-cryptographic assumption that we need to make to prove our scheme secure.

Without going into details, basic attestation is defined as a tuple of algorithms: (aBSetup, aBAttest, aBVerif), such that aBSetup(1^λ) generates some public parameters pparam from a security parameter λ; aBAttest(pparam) outputs an attestation quote denoted quote, which is set to $\aleph$ if the component is compromised; and aBVerif(quote, pparam) which outputs either 1 (the attestation is verified) or 0 (it is not). We considered a single party, which, for the purposes of the security game, may be corrupted or honest. A basic attestation scheme is secure if, once corrupted, the component is no longer able to attest itself (*i.e.*, if the component is corrupted, the only attestation quotes we can produce are $\aleph$).

We assume that this component exists, and behaves as in Figure 4.3.

4.2.1 Formalization

We consider an environment parametrized by a security parameter λ, in which we have a single party P. This party keeps track of a single *attribute*, namely a *compromise bit* γ originally set to 0. Once this bit is flipped to 1, it can never go back to 0. We denote by $P.\gamma$ the compromise bit γ of party P. We define a primitive BasicAtt as a tuple of algorithms (aBSetup, aBAttest, aBVerif):

> aBSetup$(1^\lambda) \rightarrow$ pparam: on input the security 1^λ (in unary), this algorithm outputs some public parameters pparam.
>
> aBAttest(pparam) $\rightarrow$ quote: on input the public parameters pparam, if $P.\gamma = 0$, then this algorithm outputs an attestation quote quote $\neq \aleph$ for P, and if $P.\gamma = 1$, then it outputs $\aleph$.
>
> aBVerif(pparam, (quote $\cup \aleph$)) $\rightarrow 0 \cup 1$: on input public parameters pparam and a value that is either a quote denoted quote or a special symbol $\aleph$, this algorithm outputs a bit. By convention, an output of 0 means the attestation fails, while if the

Target T		**Appraiser**
Setup phase: $\texttt{aBSetup}(1^\lambda) \rightarrow \mathsf{pparam}$		
$\texttt{aBAttest}(\mathsf{pparam}) \rightarrow \mathsf{quote}$		
	$\xrightarrow{\mathsf{quote}}$	
		$\texttt{aBVerif}(\mathsf{pparam}, \mathsf{quote})$ $\rightarrow 0$ if T compromised ($T.\gamma = 1$) $\rightarrow 1$ if T uncompromised ($T.\gamma = 0$)

Figure 4.3 *Basic attestation description with an honestly-generated target. Notice that there is no authentication involved.*

output is 1, the attestation succeeds. We require by construction that for all pparam: $\texttt{aBVerif}(\cdot, \cdot, \aleph) = 0$.

This primitive is also depicted in Figure 4.3.

We assume that, given $\mathsf{pparam} \leftarrow \texttt{aBSetup}(1^\lambda)$, if $P.\gamma = 0$ (the party is uncompromised) and $\mathsf{quote} \leftarrow \texttt{aBAttest}(\mathsf{pparam})$, then $\texttt{aBVerif}(\mathsf{pparam}, \mathsf{quote}) = 1$. Notably, we assume *perfect correctness*.

4.2.2 Security

The only security we demand from this primitive is that, if a party is compromised, then its attestation will always fail. This will happen by construction (since this is an assumed primitive) and is embedded in the security model. The adversary $\mathcal{A}$ will play a game against a challenger C. Initially, the challenger sets the system up by running aBSetup to output pparam which is given to $\mathcal{A}$. The unique party is generated, such that its corrupt bit is set to 1 ($P.\gamma = 0$).

Since $\mathcal{A}$ now has pparam, it can now run the aBAttest and aBVerif algorithms. In addition, it has access to the oBAttest oracle: $\mathsf{oBAttest}() \rightarrow (\mathsf{quote} \cup \aleph)$. This oracle calls the aBAttest() algorithm for the (corrupted) party P and returns the output to the adversary $\mathcal{A}$. The challenger stores the result in a database DB. The adversary wins if, and only if, there exists a quote in DB (possibly with $\mathsf{quote} = \aleph$) such that $\texttt{aBVerif}(\mathsf{pparam}, \mathsf{quote}) = 1$. Note that by construction our basic attestation primitive is secure, since once the compromise bit is set, the output is $\aleph$, which always yields $\texttt{aBVerif}(\mathsf{pparam}, \aleph) = 0$.

Basic attestation in reality. One may wonder at this point what our purpose might be in constructing a security model for a primitive that is by definition correct and secure. We need that security model in our reductions: we will use the attestation primitive to build stronger, linked attestation, and then we will want to make the argument that if an attacker can break the larger primitive, it will also break the smaller primitive. As the smaller primitive is secure by design, this is not possible, and hence, the larger primitive is also secure.

4.3 Authenticated Attestation

Basic attestation acts as a foolproof way of telling whether a device is compromised or not. However, the security it provides is very weak. For one thing, it has no authentication guarantees, so potentially one could use a quote that was honestly generated for an honest component to attest a compromised one. Another problem that is more subtle concerns the way components are compromised. Because the basic quotes described in the previous section have no timestamp, nor specific freshness, we cannot take into account adaptive tampering. In the security notion, the party generating the quote is either honest or compromised from the beginning. Yet, ideally we would like a primitive that ensures that a party can start out as honest (and all the quotes generated at that time verify as correct), and later be compromised (and all the quotes generated after that moment will fail). We can do this by deploying cryptographic solutions.

A relevant question is why we did not include these security aspects in the basic attestation primitive considered above. To answer this, recall that we have constructed the basic attestation tool to be secure by design. As such, it is an assumption, rather than a solution. If we also assume authentication, it would go against the principle of using minimal assumptions.

Intuitively, the security we require for this primitive will be that a valid authenticated quote for a party P and fresh auxiliary information (used as nonce) is hard to forge by an adversary which knows all the the public information, can register and compromise users, and query an attestation oracle that returns a valid quote or $\aleph$. In particular, in a secure scheme,verification should fail if either the authentication or the attestation fails, as depicted in Figure 4.4.

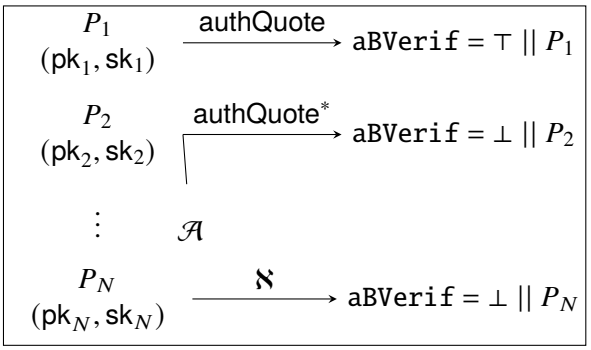


Figure 4.4 *Authenticated attestation primitive.*

4.3.1 Correctness

The correctness of our construction depends on the detection of a compromised component. There are three cases to consider:

Case 1: assume that the component is compromised. So the output attestation is $\aleph$. The component can try to authenticate this quote, but the verification will fail.

Case 2: the component (VM or hypervisor) is not compromised, and so will receive a valid attestation quote, authenticated by the TPM. This authenticated quote will verify.

Case 3: the component is not compromised, and receives a valid authentication quote. At this point, the adversary might try to forward the authenticated quote and pass it off as someone else's attestation, but this will occur with negligible probability as long as the authentication primitive is EUF-CMA secure.

4.3.2 Formalization

We consider an environment parametrized by a security parameter λ, which will contain up to $\mathsf{n}_{\mathcal{P}}$ parties. Parties keep track of two attributes: the compromise bit γ used also for basic attestation, and a set of public and secret keys $(\mathsf{pk}, \mathsf{sk})$: a public key pk (assumed to be unique per party, and known to all other parties including the adversary) and a private key sk known only to that party. We use $P.\mathsf{pk}/P.\mathsf{sk}$ to indicate the public/secret key of party P.

We define the primitive $\mathtt{AuthAtt}=(\mathtt{aAuthSetup}, \mathtt{aAuthKGen}, \mathtt{aAuthAttest}, \mathtt{aAuthVerif})$ as a tuple of algorithms: together with an auxiliary input space $\mathcal{AUX}$ as follows:

- $\mathtt{aAuthSetup}(1^\lambda) \rightarrow \mathsf{pparam}$: on input the security 1^λ (in unary), this algorithm outputs some public parameters pparam. This value also includes the number $\mathsf{n}_{\mathcal{P}}$, which is a function of λ.

- $\mathtt{aAuthKGen}(P) \rightarrow (P.\mathsf{pk}, P.\mathsf{sk})$: on input a party identifier P, this stateful algorithm checks whether P has already been registered; if not, it outputs a tuple of public/private keys $(\mathsf{pk}, \mathsf{sk})$. These values are stored by the keys attribute of P.

- $\mathtt{aAuthAttest}(\mathsf{pparam}, P.\mathsf{sk}, \mathsf{AD}) \rightarrow \mathsf{authQuote} \cup \aleph$: on input the public parameters pparam, a private key $P.\mathsf{sk}$ of a party P, and a value $\mathsf{AD} \in \mathcal{AUX}$, this algorithm outputs either an authenticated quote $\mathsf{authQuote}$ or a special failure symbol $\aleph$.

- $\mathtt{aAuthVerif}(\mathsf{pparam}, P.\mathsf{pk}, \mathsf{AD}, (\mathsf{authQuote} \cup \aleph)) \rightarrow 0 \cup 1$: on input public parameters pparam, a public key $P.\mathsf{pk}$ of a party P, an auxiliary value $\mathsf{AD} \in \mathcal{AUX}$, and a value that is either a quote $\mathsf{authQuote}$ or a special symbol $\aleph$, this algorithm outputs a bit. By convention, an output of 0 means the attestation fails, while if the output is 1, the attestation succeeds.

We require that this primitive be *correct* in two ways: $\mathtt{aAuthVerif}(\cdot, \cdot, \cdot, \aleph) = 0$ for all possible input values in the first three parameters, and: for all $\mathsf{pparam} \leftarrow \mathtt{aAuthSetup}(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{aAuthKGen}(P)$, $\mathsf{AD} \in \mathcal{AUX}$, if $\mathsf{authQuote} \leftarrow \mathtt{aAuthAttest}(\mathsf{pparam}, P.\mathsf{sk}, \mathsf{AD})$ and $\mathsf{authQuote} \neq \aleph$, $\mathtt{aAuthVerif}(\mathsf{pparam}, P.\mathsf{pk}, \mathsf{AD}, \mathsf{authQuote}) = 1$.

4.3.3 Construction

We construct an authenticated attestation scheme out of basic authentication, a large set of nonces $\mathcal{N} := \{0,1\}^\ell$ (with ℓ chosen as a function of the security parameter λ), and an EUF-CMA-secure signature scheme $\mathtt{Sig} = (\mathtt{aSigKGen}, \mathtt{aSigSign}, \mathtt{aSigVerif})$. We thus instantiate $\mathcal{AUX} := \mathcal{N}$, and our $\mathtt{AuthAtt}$ scheme is as follows:

Target T		**Appraiser**
Setup phase: $\texttt{aAuthSetup}(1^\lambda) \to \mathsf{pparam}$		
$\texttt{aAuthKGen} \to (T.\mathsf{pk}, T.\mathsf{sk})$		
$\texttt{aAuthAttest}(\mathsf{pparam}, T.\mathsf{sk}, \mathsf{AD}) \to (\mathsf{quote}, \sigma)$	$\xrightarrow{\mathsf{authQuote}=(\mathsf{quote},\sigma)}$	
		$\texttt{aAuthVerif}(\mathsf{pparam}, T.\mathsf{pk}, \mathsf{AD}, (\mathsf{quote}, \sigma))$ $\to 0$ if T compromised ($\mathsf{authQuote} = \aleph$ or σ invalid) $\to 1$ if T uncompromised ($\mathsf{authQuote} \neq \aleph$ and σ valid)

Figure 4.5 *Authenticated attestation description built upon basic attestation (Figure 4.3) where target T is a party generated honestly (which can be compromised later) and the verifier is an appraiser measuring T.*

- $\texttt{aAuthSetup}(1^\lambda) \to \mathsf{pparam}$: this algorithm runs $\texttt{aBSetup}(1^\lambda)$ a number $\mathsf{n}_\mathcal{P}$ of times, outputting $\mathsf{pparam}_1, \mathsf{pparam}_2, \ldots, \mathsf{pparam}_{\mathsf{n}_\mathcal{P}}$. Each time pparam_i is created, a party handle P_i is also created (it will be the party associated with the instance of `BasicAtt` run for those parameters). It sets $\mathsf{pparam} := (\mathsf{pparam}_1, \mathsf{pparam}_2, \ldots, \mathsf{pparam}_{\mathsf{n}_\mathcal{P}}, \mathsf{n}_\mathcal{P})$, and outputs this value.

- $\texttt{aAuthKGen}(P_i) \to (P_i.\mathsf{pk}, P_i.\mathsf{sk})$: it keeps a counter (starting from 0), which indicates how many times this algorithm has been run. If at the time this algorithm is queried $\mathsf{counter} < \mathsf{n}_\mathcal{P}$, then `aAuthKGen` runs `aSigKGen` as a black box and outputs the resulting $(\mathsf{pk}, \mathsf{sk})$ (public and private) keys. It sets $P_i.\mathsf{pk} := \mathsf{pk}$ and $P_i.\mathsf{sk} := \mathsf{sk}$. Party P_i is then initialized with these keys.

- $\texttt{aAuthAttest}(\mathsf{pparam}, P.\mathsf{sk}, R) \to \mathsf{authQuote} \cup \aleph$: on input the public parameters pparam, a private key $P.\mathsf{sk}$ of a party P (which has already been registered), and a value $R \xleftarrow{\$} \mathcal{N}$, this algorithm first runs $\mathsf{quote} \leftarrow \texttt{aBAttest}(\mathsf{pparam})$, then the algorithm signs $\sigma \leftarrow \texttt{aSigSign}(P.\mathsf{sk}, (\mathsf{quote}, R))$, that is, it signs a concatenation of the nonce and the obtained quote. The output of this algorithm is $\mathsf{authQuote} := (\mathsf{quote}, \sigma)$. If the required party or key does not exist, the value $\aleph$ is output by default. If $\mathsf{quote} = \aleph$, then we instantiate $\mathsf{authQuote} = \aleph$.

- $\texttt{aAuthVerif}(\mathsf{pparam}, P.\mathsf{pk}, R, (\mathsf{authQuote} \cup \aleph)) \to 0 \cup 1$: on input public parameters pparam, a public key $P.\mathsf{pk}$ of a party P, an auxiliary value $R \in \mathcal{N}$, this algorithm first checks if the last input is $\aleph$; if so, the algorithm outputs 0 by default. Else, the algorithm parses $\mathsf{authQuote} = (\mathsf{quote}, \sigma)$ (with $\mathsf{quote} \neq \aleph$ by construction), then runs $b \leftarrow \texttt{aSigVerif}(P.\mathsf{pk}, \mathsf{quote}, \sigma)$ and $d \leftarrow \texttt{aBVerif}(\mathsf{pparam}, \mathsf{quote})$. The algorithm outputs $b \wedge d$. Notably, 1 is output if, and only if, signature and basic attestation are valid concomitantly.

We claim that this scheme, which is also depicted in Figure 4.5 has the correctness and security properties required, given that ℓ is large, that the basic attestation scheme achieves the notion of security in Section 4.2, and that the signature scheme is EUF-CMA-secure.

pparam $\leftarrow$ `aAuthSetup`(1^λ) retrieve N from pparam; counter $\leftarrow 1$ $\mathcal{L}_{\text{sign}} \leftarrow \emptyset$ abort if $1 \leftarrow$ `aBVerif`(pparam, quote*) abort if $1 \leftarrow$ `aSigVerif`(P^*.pk, quote*, σ^*) $(P^*, \text{AD}^*, \text{authQuote}^*) \leftarrow \mathcal{A}^O(\text{pparam})$
$\mathcal{A}$ wins iff.: `aAuthVerif`(pparam, P^*, AD*, authQuote*) = 1 AND $(P^*, \text{AD}^*, \text{authQuote}^*) \notin \mathcal{L}_{\text{sign}}$

Figure 4.6 *The experiment* `AuthSec` *with oracles* $O = \{\text{oBAttest}(), \text{oAuthReg}(.), \text{oAuthAttest}(.), \text{oCompromise}(.)\}$. *Recall that* authQuote $= (\text{quote}^*, \sigma^*)$. *We depict with grey boxes the changes made in the sequence of games.*

4.3.4 Security

Formally, $\mathcal{A}$ will play the `AuthSec` game against a challenger C, which begins the game by running `aAuthSetup` and outputting pparam to $\mathcal{A}$. The challenger also initializes $\text{n}_\mathcal{P}$ to 1. The adversary then has access to the following oracles:

- oAuthReg() $\rightarrow (P_i, P_i.\text{pk})$: if $i \leq \text{n}_\mathcal{P}$, it runs `aAuthKGen`$(P_i) \rightarrow (P_i.\text{pk}, P_i.\text{sk})$. It outputs P_i.pk to all parties and keeps P_i.sk private, stored in the keys attribute of party P_i. A handle (in practice the index) of this party is also returned to $\mathcal{A}$.

- oAuthAttest$(P_i, \text{AD}) \rightarrow \text{authQuote} \cup \aleph$: this oracle runs the `aAuthAttest` algorithm on pparam, P_i.sk and input AD, and returns the output. On adversarially chosen input P_i and AD, the oracle updates a list $\mathcal{L}_{\text{sign}} \leftarrow \mathcal{L}_{\text{sign}} \cup (P_i, \text{AD}, \text{authQuote})$.

- oCompromise$(P_i) \rightarrow$ OK : this oracle allows an adversary to compromise party P_i, thus changing $P_i.\gamma$ to 1.

- oAuth$(P_i, M) \rightarrow \sigma_M$: this oracle can only be queried for a party whose compromise bit is 1, and it outputs an EUF-CMA-secure signature keyed with P_i.sk on a message M. We require that M be outside the range of any basic attestation scheme. This last oracle reflects the fact that compromised parties can access a signing function within the TPM.

Finally the adversary outputs a tuple $(P, \text{AD}, \text{authQuote})$. It is said to *win* if and only if the following condition holds: `aAuthVerif`(pparam, P, AD, authQuote) = 1 and there exist no tuples $(P, \text{AD}, \text{authQuote})$ such that $(\text{authQuote} \neq \aleph) \rightarrow \text{oAuthAttest}(\text{pparam}, P, \text{AD})$ for the current public parameters pparam (output by the challenger C).

Theorem 7 (Secure Authenticated Attestation) *The* `AuthAtt` *scheme is secure assuming that (1)* `BasicAtt` *scheme is secure (2) the size of* $\mathcal{N}$ *is large and (3) the* `Sig` *signature scheme is EUF-CMA secure.*

4.3.5 Security Proof

Let $\mathcal{A}$ be a probabilistic polynomial-time algorithm. The goal of $\mathcal{A}$ is to provide a signed quote with correct auxiliary value (a nonce) such that the quote and the signature are valid for a fresh nonce. Note that $\mathcal{A}$ has access to oracles as depicted in Figure 4.6.

We propose a proof using a sequence of game hops as introduced in [Sho06]. The initial game corresponds to the security game `AuthSec`. The successive games are slight modification to its previous one to end up with a game corresponding to generic primitive game (such as EUF-CMA). The goal of the proof is to show that $\mathcal{A}$ has negligible probability of winning `AuthSec`.

$\mathbb{G}_0$: This is the original security game given in Figure 4.6.

$\mathbb{G}_1$ (transition based on indistinguishability): This game is defined as the previous one except that the challenger aborts the game if a compromised component is able to generate a valid attestation quote. Suppose that $\mathcal{A}$ has a non-negligible advantage ϵ_B of winning the basic attestation game. This means that there exists a party P such that $P.\gamma = 1$ (*i.e.*, P is compromised) but $\mathsf{quote} \in \mathsf{DB}$ with $\mathtt{aBVerif}(\mathsf{pparam}, \mathsf{quote}) = 1$ (note that $\mathsf{quote} = \aleph$ potentially). By difference lemma we have:

$$|\Pr[\mathcal{A} \text{ wins } G_0] - \Pr[\mathcal{A} \text{ wins } G_1]| \leq \varepsilon_B.$$

$\mathbb{G}_2$: This game is defined as the previous one except that the game aborts if $\mathcal{A}$ can generate a valid signature. We show that:

$$|\Pr[\mathcal{A} \text{ wins } G_1] - \Pr[\mathcal{A} \text{ wins } G_2]| = \frac{\varepsilon_{\text{EUF-CMA}}}{N}$$

where $\varepsilon_{\text{EUF-CMA}}$ is the advantage of EUF-CMA security game. The proof is done by reduction.

Assume that $\mathcal{A}$ can generate a valid $\mathsf{authQuote}^*$, *i.e.*, $\mathtt{aAuthVerif}(\mathsf{pparam}, P^*, \mathsf{AD}^*, \mathsf{authQuote}^*) = 1$ with $(P^*, \mathsf{AD}^*, \mathsf{authQuote}^*) \notin \mathcal{L}_{\mathsf{sign}}$. We then show that there exists adversary $\mathcal{B}$ using $\mathcal{A}$ as a sub-routine with non-negligible advantage of winning the EUF-CMA security game.

Adversary $\mathcal{B}$ simulates the game of $\mathcal{A}$ thus acting as the challenger in the `AuthAtt` game. The behavior of $\mathcal{B}$ is defined as follows:

- receives pk from its own challenger of the EUF-CMA game.
- runs the `aAuthSetup` algorithm to get pparam (and also N).
- randomly selects $i^* \xleftarrow{\$} \{1, \ldots, N\}$. Two cases need to be studied depending on the i^{th}query of $\mathcal{A}$:
  - $i \neq i^*$. When $\mathcal{A}$ calls oracle $\mathsf{oAuthReg}()$ then $\mathcal{B}$ runs $\mathtt{aAuthKGen}(P_i)$ to retrieve $(P_i.\mathsf{pk}, P_i.\mathsf{sk})$. $\mathcal{B}$ sends back $P_i.\mathsf{pk}$ to $\mathcal{A}$. When $\mathcal{A}$ calls oracle $\mathsf{oAuthAttest}()$ then $\mathcal{B}$ runs algorithm `aAuthAttest` (which is possible since $\mathcal{B}$ has the corresponding secret key). $\mathcal{B}$ sends back to $\mathcal{A}$ the output of the algorithm. Note that

in this case, the simulation is the same as the original game since $\mathcal{B}$ uses the same algorithm of the oracles.

– $i = i^*$. In this case, $\mathcal{B}$ will inject its own material to use it in its EUF-CMA game. When $\mathcal{A}$ calls oracle $\mathsf{oAuthReg}(P_{i^*})$ then $\mathcal{B}$ simply returns pk. Note that in this case, $\mathcal{B}$ does not have access to sk. Thus when $\mathcal{A}$ calls oracle $\mathsf{oAuthAttest}()$, $\mathcal{B}$ cannot sign the quote. Instead, $\mathcal{B}$ runs $\mathtt{aBAttest}(\mathsf{pparam}_{i^*})$ and sends to its challenger $\mathsf{quote}||\mathsf{AD}$. In response, its challenger will send a signed value of it using sk. $\mathcal{B}$ then forward this to $\mathcal{A}$. The view of $\mathcal{A}$ that is different from the original game in this case is the output of the $\mathsf{oAuthAttest}()$ oracle. The latter runs algorithm $\mathtt{aAuthAttest}$ which generate a quote $\mathsf{quote} \leftarrow \mathtt{aBAttest}(\mathsf{pparam}_i)$ and a signature $\sigma \leftarrow \mathtt{aSigSign}(P_i.\mathsf{sk}, (\mathsf{quote}, \mathsf{AD}))$. In the simulation, $\mathcal{B}$ has access to algorithm $\mathtt{aBAttest}()$ but the signature scheme is different. Yet, both signature schemes are EUF-CMA thus their outputs are indistinguishable (meaning that $\mathcal{A}$ cannot decide from which schemes the output comes from with non-negligible probability) since the keys have the same probability distribution. Hence, the simulation of the game by $\mathcal{B}$ and the real game are indistinguishable.

- When $\mathcal{A}$ returns its forgery on query i, $\mathcal{B}$ parses $\mathsf{authQuote}_i$ as $\mathsf{authQuote}_i := (m^*||\mathsf{AD}^*, \sigma^*)$.
- Finally $\mathcal{B}$ returns $(m^*||\mathsf{AD}^*, \sigma^*)$ to its challenger and wins if $\mathcal{A}$ forges the i^*query (meaning that $i = i^*$).

By combining the results, we have:

$$\Pr[\mathcal{A} \text{ wins } G_0] \leq \varepsilon_B + \frac{\varepsilon_{\text{EUF-CMA}}}{N}$$

which is negligible.

4.4 Linked Attestation

Authenticated attestation allows the attestation of one (out of many) components, based on that component's unique secret key. If we define now parties as being either VMs or hypervisors, the notion of authenticated attestation suffices to capture the basic guarantees of multi-channel deep-attestation. However, in this paper our goal is to allow parties to *link* their attestations (a hypervisor's attestation should, *e.g.*,, be linkable to various VMs hosted on that platform).

The linked attestation takes place in an environment where several parties are registered in a linked way – this corresponds to a single platform. A first step is platform registration, by which several parties are linked on the same underlying hardware. Each entity later generates a linkable attestation – verifiable on its own, and linkable with other linkable attestations.

Although our application scenario is that of linking VM and hypervisor attestations, we make our framework more generic than that. Instead of just two types of components, we consider linkable sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_L$, which resemble equivalence classes. These sets are

$\underline{\texttt{aLSetup}}(1^\lambda)$:
$\mathsf{pparam}' \leftarrow \texttt{aAuthSetup}(1^\lambda)$
Return $\mathsf{pparam} \leftarrow \mathsf{pparam}'$

// Registers a platform with set $\mathbf{s}_1$ of VMs and the hypervisor in $\mathbf{s}_2$
$\underline{\texttt{aLReg}}(\mathbf{s}_1, \mathbf{s}_2)$: For each $i \in \{1, 2\}$:
    For each $j \in \mathbf{s}_i$:
        $(P_j.\mathsf{pk}, P_j.\mathsf{sk}) \leftarrow \texttt{aAuthKGen}(P_j)$
    Group all $P_j.\mathsf{pk}$ into PK_i and all $P_j.\mathsf{sk}$ into SK_i
Return$\{(\mathsf{PK}_1, \mathsf{SK}_1), (\mathsf{PK}_2, \mathsf{SK}_2)\}$

// Attesting VM P on platform $(\mathbf{s}_1, \mathbf{s}_2)$ for nonce AD
$\underline{\texttt{aLAttest}}(\mathsf{pparam}, \mathcal{PK}, P.\mathsf{sk}, (\mathbf{s}_1, \mathbf{s}_2), \mathsf{AD})$:
Get $P.\mathsf{pk}$ matching $P.\mathsf{sk}$ from $\mathcal{PK}$
$\mathsf{lkaux} \leftarrow P.\mathsf{pk}$ *// The linking information is P's public key*
$\mathsf{AD}^* \leftarrow H(\mathsf{AD}||\mathsf{lkaux})$ *// Embed* lkaux *into attestation nonce*
$\mathsf{authQuote} \leftarrow \texttt{aAuthAttest}(\mathsf{pparam}', P.\mathsf{sk}, \mathsf{AD}^*)$
$\mathsf{linkedQuote} \leftarrow \mathsf{authQuote}$
Return $(\mathsf{linkedQuote}, \mathsf{lkaux})$

$\underline{\texttt{aLVerif}}(\mathsf{pparam}, P.\mathsf{pk}, \mathsf{linkedQuote}, \mathsf{AD}, \mathsf{lkaux})$: *// Verify attestation quote of party P*
$\mathsf{AD}^* \leftarrow H(\mathsf{AD}||\mathsf{lkaux})$;
$\mathsf{authQuote} \leftarrow \mathsf{linkedQuote}$
Return $\texttt{aAuthVerif}(\mathsf{pparam}', P.\mathsf{pk}, \mathsf{authQuote}, \mathsf{AD}^*)$

// Attest hypervisor P on platform $(\mathbf{s}_1, \mathbf{s}_2)$ with nonce AD
$\underline{\texttt{aLAttest}}(\mathsf{pparam}, \mathcal{PK}, P.\mathsf{sk}, (\mathbf{s}_1, \mathbf{s}_2), \mathsf{AD})$:
Parse $\mathcal{PK}$ as $\mathcal{PK}[1], \mathcal{PK}[2]$ *// $\mathcal{PK}[1]$ is the set of all VM pks*
Parse $\mathcal{PK}[1]$ as $\mathsf{PK}^1, \mathsf{PK}^2 \ldots \mathsf{PK}^{|\mathcal{S}_1|}$ *// PK^i contains the keys of all VMs on platform i*
Set $\mathsf{lkaux} \leftarrow \mathsf{PK}^k$ with k the index of $\mathbf{s}_1$ in $\mathcal{S}_1$ *// lkaux is now the list of all VM keys*
$\mathsf{AD}^* \leftarrow H(\mathsf{AD}||\mathsf{lkaux})$ *// Embed* lkaux *into a new attestation nonce*
$\mathsf{authQuote} \leftarrow \texttt{aAuthAttest}(\mathsf{pparam}', P.\mathsf{sk}, \mathsf{AD}^*)$
$\mathsf{linkedQuote} \leftarrow \mathsf{authQuote}$
Return $(\mathsf{linkedQuote}, \mathsf{lkaux})$

$\underline{\texttt{aLLink}}(\mathsf{pparam}, \mathcal{PK}, \Pi_1, \Pi_2)$: *// Link VM quotes from Π_1 and hypervisor quote from Π_2*
Initialize $\mathsf{AUX}_{\mathsf{vm}} \leftarrow \emptyset$
For each $(P_j.\mathsf{pk}, \mathsf{AD}, \mathsf{linkedQuote}, \mathsf{lkaux}) \in \Pi_1$:
    Return 0 if $\texttt{aLVerif}(\mathsf{pparam}', P_j.\mathsf{pk}, \mathsf{linkedQuote}, \mathsf{AD}, \mathsf{lkaux})$ return 0
    Return 0 if $\mathsf{lkaux} \neq P_j.\mathsf{pk}$ *// Linking fails if quotes fail to verify or authenticate each VM*
    Add lkaux to $\mathsf{AUX}_{\mathsf{vm}}$ *// Each* lkaux *here is a VM public key.*
Parse Π_2 as $(P_j.\mathsf{pk}, \mathsf{AD}, \mathsf{linkedQuote}, \mathsf{lkaux})$
If $\texttt{aLVerif}(\mathsf{pparam}', P_j.\mathsf{pk}, \mathsf{linkedQuote}, \mathsf{lkaux})$ return 0
$\mathsf{AUX}_{\mathsf{hym}} \leftarrow \mathsf{lkaux}$ *// This* lkaux *is a list of VM public keys.*
Return 0 if $\mathsf{AUX}_{\mathsf{vm}}$ is not a subset of $\mathsf{AUX}_{\mathsf{hym}}$
Return 1

Figure 4.7 *Our linked attestation scheme for platforms with 2 types of components: VMs (stored in $\mathcal{S}_1$) and hypervisors (stored in $\mathcal{S}_2$). Each type of component attests via a different* aLAttest *algorithm, the main difference between them being that the hypervisor embeds a list of public keys in its nonce.*

defined such that any party in one set (say $P_{\mathcal{S}_1}$) can produce an attestation that is linked to attestations produced by parties in sets $\mathcal{S}_2, \ldots, \mathcal{S}_L$. We write $P \diamond Q$ to say that two parties are linked. The relation is reflexive ($P \diamond P$), symmetric (if $P \diamond Q$, then $Q \diamond P$), and transitive (if $P \diamond Q$ and $Q \diamond R$, then $P \diamond R$). An intuitive depiction of these sets appears in Figure 4.8.

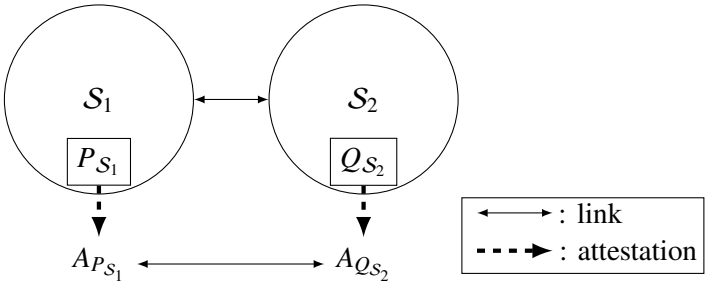


Figure 4.8 *The sets $\mathcal{S}_1, \mathcal{S}_2$ contain parties (e.g., VM and attestation server). In this example, $P_{\mathcal{S}_1}$ lies in $\mathcal{S}_1$ and $Q_{\mathcal{S}_2}$ lies in $\mathcal{S}_2$. There can exist links between the sets and also between parties' attestation. In this example, P and Q are linked (denoted $P \diamond Q$).*

We formalize a linked-attestation scheme `LinkedAtt` as a tuple of algorithms `LinkedAtt` = (`aLSetup`, `aLReg`, `aLAttest`, `aLVerif`, `aLLink`), defined for some auxiliary set $\mathcal{AUX}$. We illustrate the syntax in Figure 4.9.

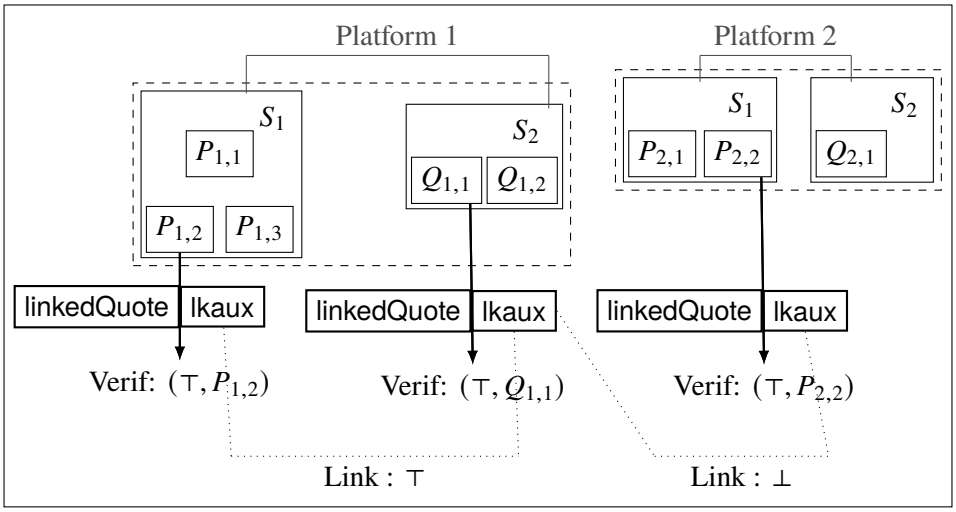


Figure 4.9 *Linked attestation primitive. The dashed line indicates a platform under the same registration. In this example, both platforms are composed of two subsets (namely S_1 and S_2). There are a total of three quote verifications ($P_{1,2}, Q_{1,1}, P_{2,2}$). The link verification outputs true when the devices are registered under the same platform and false otherwise.*

- $\texttt{aLSetup}(1^\lambda) \rightarrow \mathsf{pparam}$: on input the security parameter 1^λ (in unary), this algorithm outputs public parameters pparam. This security parameter includes the maximal number of allowed disjoint linkable sets, which we denote as L.

- $\texttt{aLReg}(\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_L) \rightarrow \{(\mathsf{PK}_1, \mathsf{SK}_1), \ldots (\mathsf{PK}_L, \mathsf{SK}_L)\}$: this algorithm keeps as state a number L of sets $\mathcal{S}_i$ originally set to $\emptyset$, and a vector of sets of public keys $\mathcal{PK}$ (also

initialized to $\emptyset$). On input a number of subsets $\mathbf{s}_i$ ($i = 1, 2, \ldots, L$), this algorithm first checks that $\forall i, j$, $\mathbf{s}_i \cap \mathcal{S}_j = \emptyset$ (else the algorithm outputs $\perp$). If the relation is true, then the algorithm generates for each party $P_j \in \mathbf{s}_i$, (for all j, i) a tuple of public and private keys $P_j.\mathsf{pk}, P_j.\mathsf{sk}$, initializing P_j with those keys. We require the uniqueness of all the generated public keys. The subsets $\mathbf{s}_i$ are each added to greater sets $\mathcal{S}_i$. The algorithm groups the keys of all parties $P_j \in \mathbf{s}_i$ in a pair of private/public key subsets: $(\mathsf{PK}_i, \mathsf{SK}_i)$, updating the i-th component $\mathcal{PK}[i]$ of $\mathcal{PK}$ as $\mathcal{PK}[i] \cup \mathsf{PK}_i$. All parties are given access to the public-key subsets (and more generally, to $\mathcal{PK}$).

- $\mathtt{aLAttest}(\mathsf{pparam}, \mathcal{PK}, P.\mathsf{sk}, (\mathbf{s}_1, \ldots, \mathbf{s}_L), \mathsf{AD}) \rightarrow (\mathsf{linkedQuote} \cup \aleph, \mathsf{lkaux})$: on input public parameters pparam, the current set of public keys $\mathcal{PK}$, the private key $P.\mathsf{sk}$ of some party P, subsets $\mathbf{s}_i \in \mathcal{S}_i$, and an auxiliary value $\mathsf{AD} \in \mathcal{AUX}$, this algorithm outputs either a linked quote $\mathsf{linkedQuote}$ or a special failure symbol $\aleph$, and a different value lkaux (this last entry could be used to store linkage-related information).

- $\mathtt{aLVerif}(\mathsf{pparam}, P.\mathsf{pk}, (\mathsf{linkedQuote} \cup \aleph), \mathsf{AD}) \rightarrow 0 \cup 1$: On input the public parameters pparam, a public key $P.\mathsf{pk}$, a linked quote (or a failure symbol $\aleph$), and an auxiliary value AD, this algorithm outputs a verification bit. By convention, 0 means failure and 1 means acceptance of the attestation.

- $\mathtt{aLLink}(\mathsf{pparam}, \mathcal{PK}, \mathrm{LI}_1, \ldots, \mathrm{LI}_L) \rightarrow 0 \cup 1$: on input the public parameters pparam, the set of public keys $\mathcal{PK}$, and subsets LI_i containing elements of the form $(P_j.\mathsf{pk}, \mathsf{AD}, (\mathsf{linkedQuote} \cup \aleph), \mathsf{lkaux})$, this algorithm outputs 1 if the quotes in all the indicated subsets can all be linked (thus also indicating the parties are linked) or 0 otherwise.

By convention, we allow the use of $\emptyset$ to indicate that any of the input or output (sub)sets to also be empty.

The security of linked attestation informally states that an adversary, which has Person-in-the-Middle capabilities and can compromise devices at will, cannot make it appear that two devices are linked when they are not, in fact, so.

A significant limitation on the adversary's capabilities is that compromising a device will not leak its private keys (which are assumed to be held by a TPM). However, the adversary will gain a limited oracle access to those keys upon compromising the device. The limitations to those queries follow rules of access to an actual TPM.

4.4.1 Construction

We provide a construction for platforms that have two types of components: virtual machines (VMs) and their managing hypervisor. Thus, in our instantiation, $L = 2$. We use an authenticated attestation scheme ($\mathtt{aAuthSetup}$, $\mathtt{aAuthKGen}$, $\mathtt{aAuthAttest}$, $\mathtt{aAuthVerif}$) as a black box. The basic construction is depicted in Figure 4.7. During setup, our linked-attestation scheme first runs $\mathtt{aAuthSetup}$ and outputs pparam and $L = 2$. Note that by construction $\mathtt{aAuthSetup}$ must output a number $\mathsf{n}_{\mathcal{P}}$, denoting the maximal number of parties that can be set up. This counter will represent a global maximum to parties of all

types that will exist in our ecosystem. Following setup, one can register a subset of VMs together with a hypervisor. The algorithm runs the key-generation algorithm `aAuthKGen` of the underlying authenticated attestation scheme for each party, independently (note that this also ensures that the total number of parties remains at most $\mathsf{n}_{\mathcal{P}}$). Finally, keys are grouped by types of parties: keys of VMs are output in a set of public keys PK_1 and the key of the hypervisor is output as PK_2.

The VMs and hypervisor generate linked attestations differently. The hypervisor first fetches the public keys of all the components registered with it on the same platform. It computes a new nonce as the hash of two concatenated values: the original auxiliary value AD and the list of the public keys. The component then runs `aAuthAttest` on the public parameters, this new nonce, and its private key, outputting the authenticated quote. By contrast, when a VM attests, it computes a new nonce from the original auxiliary value AD and (only) its own public key. The authenticated quote is provided as the VM's linked quote.

A VM (or a set of VMs) are considered to be linked to a hypervisor if, and only if, the following conditions hold simultaneously: (1) the attestations of all the purportedly-linked parties verify individually (if we run `aAuthVerif` it returns 1 for each individual attestation); (2) the public key that was successfully used to verify each of the VMs' attestation is part of the auxiliary value lkaux forwarded by the hypervisor.

4.4.2 Correctness

The `LinkedAtt` scheme is built upon the `AuthAtt` scheme. There are two types of component to consider, VM and hypervisor. When a component is registered on a platform, its public key is appended in a list (PK_1 for VMs, and PK_2 for the hypervisor). The public key of a VM is appended to the quote in `aLAttest` and can be retrieved by the hypervisor. The latter can link the attestation to a public key via algorithm `aLLink`. We consider two cases to verify the correctness (1) a VM (not compromised) is not registered on the platform, and (2) a component (VM or hypervisor) is compromised. For (1) the attestation will be correct since the component is not compromised, but the linking process will abort since the public key does not belong to PK_1. For (2) if a VM (or the hypervisor) is compromised then the attestation will fail since the authenticated attestation is supposed to be correct (the `aAuthAttest` algorithm is executed to generate the quote).

Theorem 8 (Secure Linked Attestation) *The* `LinkedAtt` *scheme is secure assuming* `AuthAtt` *scheme is secure and hash function H is collision resistant.*

4.4.3 Security

We define the security of Linked Attestation as a game $\mathtt{LinkSec}_{\lambda,\mathsf{F}}$ played by an adversary $\mathcal{A}$ against its challenger C. The game is parametrized by a security parameter λ and a set of functions F, which we call the *permitted key-access functions*. The challenger begins by running $\mathtt{aLSetup}(1^{\lambda})$, returning pparam to the adversary, and then it instantiates two lists: a list of parties $\mathcal{L}_{\mathsf{Reg}} = \emptyset$ and a list of linkable attestations $\mathcal{L}_{\mathsf{sign}} = \emptyset$. The adversary then plays its game by using the following oracles adaptively:

- $\mathsf{oLReg}(n_1, \ldots n_L) \rightarrow (\mathsf{PK}_1, \ldots \mathsf{PK}_L)$: the linked user-registration oracle creates a linked platform consisting of n_i components of the type indicated by $\mathcal{S}_i$. The challenger first instantiates a counter $N_i = 0$ for all i; it also instantiates subsets $\mathbf{s}_i$ as a tuple of n_i handles $P_{i,j}$, with $N_i + 1 \leq j \leq N_i + n_i$ and then runs the algorithm $\mathtt{aLReg}(\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_L)$, instantiating the parties with their keys and outputting the public key-sets to the adversary. The subset consisting of the list of subsets is added to $\mathcal{L}_{\mathsf{Reg}}$. We note that this way of registering parties ensures by construction that no party finds itself in multiple sets, nor on multiple platforms.

- $\mathsf{oLHAttest}(P, (\mathbf{s}_1, \ldots, \mathbf{s}_L), \mathsf{AD}) \rightarrow (\mathsf{linkedQuote} \cup \aleph, \mathsf{AD}^*)$: this oracle first verifies that $P.\gamma = 1$. If the condition is false (P is compromised), then this oracle outputs an error symbol $\perp$ (compromised parties must use the oracle oLCAttest described below). If the condition is true, then this algorithm runs $\mathtt{aLAttest}(\mathsf{pparam}, \mathcal{PK}, P.\mathsf{sk}, (\mathbf{s}_1, \ldots, \mathbf{s}_L), \mathsf{AD})$, and returns the output to the adversary. The tuple $(P, (\mathbf{s}_1, \ldots, \mathbf{s}_L), \mathsf{AD}, \mathsf{linkedQuote}, \mathsf{AD}^*)$ is stored in $\mathcal{L}_{\mathsf{sign}}$.

- $\mathsf{oCompromise}(P) \rightarrow \mathsf{OK}$: this oracle allows an adversary to compromise party P, thus changing $P.\gamma$ to 1.

- $\mathsf{oLCAttest}(P, (\mathbf{s}_1, \ldots, \mathbf{s}_L), \mathsf{AD}, f) \rightarrow (\aleph, \mathsf{AD}^*)$: this oracle first checks that $P.\gamma = 1$ (else, $\perp$ is returned as an output). If the condition holds, then this oracle first checks that $f \in F$ and if so, it runs f on $P.\mathsf{sk}$ and input AD to output AD^*. Then it runs $\mathsf{oLHAttest}(P, (\mathbf{s}_1, \ldots, \mathbf{s}_L), \mathsf{AD})$ to obtain linkedQuote (the second output is discarded). Note that by the security of the linked attestation primitive, we will have that $\mathsf{linkedQuote} = \aleph$. The tuple $(P, (\mathbf{s}_1, \ldots, \mathbf{s}_L), \mathsf{AD}, \mathsf{linkedQuote}, \mathsf{AD}^*)$ is added to $\mathcal{L}_{\mathsf{sign}}$ and $(\mathsf{linkedQuote}, \mathsf{AD}^*)$ is returned to $\mathcal{A}$.

At the end of its interaction, $\mathcal{A}$ outputs a party P and a tuple of subsets $(\tilde{\mathbf{s}}_1, \ldots, \tilde{\mathbf{s}}_L)$ with an index $i^\star$ such that $\forall i \neq i^\star$, $\mathbf{s}_i := \tilde{\mathbf{s}}_i$ and $\mathbf{s}_{i^\star} := \tilde{\mathbf{s}}_{i^\star} \cup \{P\}$. In addition the adversary outputs for every party $P \in \mathbf{s}_1 \cup \cdots \cup \mathbf{s}_L$ (parties being indexed as $P_{i,j}$) a tuple $(\mathsf{AD}, \mathsf{linkedQuote}, \mathsf{AD}^*)$ such that $(\cdot, \cdot, \mathsf{AD}, \mathsf{linkedQuote}, \mathsf{AD}^*) \in \mathcal{L}_{\mathsf{sign}}$.

We say the adversary *wins* if all the following conditions hold simultaneously:

- For each $\mathbf{s}_i$ the parties inside this set are all registered *i.e.*,they were output by oLReg. In addition P is registered;

- There exists at least one party $Q \in \mathbf{s}_j$ such that P and Q were issued from different oLReg queries;

- By setting $\mathrm{II}_i := (P.\mathsf{pk}, \mathsf{AD}, (\mathsf{linkedQuote} \cup \aleph), \mathsf{AD}^*)$ and, for $k \neq i$, for all $P_{k,j} \in \mathbf{s}_j$, $\mathrm{II}_k := (P_{k,j}.\mathsf{pk}, \mathsf{AD}, (\mathsf{linkedQuote}_{k,j} \cup \aleph), \mathsf{AD}^*_{k,j})$, it holds that $\mathtt{aLLink}(\mathsf{pparam}, \mathcal{PK}, \mathrm{II}_1, \ldots, \mathrm{II}_L) = 1$.

In other words, the adversary wins if it is able to make attestations stored in $\mathcal{L}_{\mathsf{sign}}$ for parties registered on different platforms (P and Q) link. Note that there are two ways that an attestation can end up in $\mathcal{L}_{\mathsf{sign}}$: either it is issued for an honest component (and

then it should hold that $\mathsf{linkedQuote} \neq \aleph$), or it is issued for a compromised party, for an adversarially-chosen evaluation of a permitted function f on a secret key (in which case $\mathsf{linkedQuote} = \aleph$). In other words, at this point a compromised component cannot just bind the output AD^* from the function evaluation oracle with a different quote. The adversary will gain this ability at the next step (when the freshness AD will no longer be chosen by the adversary).

4.4.4 Security Proof

We will now prove our construction is secure with respect to the $\mathtt{LinkSec}_{\lambda,\mathsf{F}_{\mathsf{Sign}}}$ experiment.

$\mathbb{G}_0$: The original security game $\mathtt{LinkSec}_{\lambda,\mathsf{F}_{\mathsf{Sign}}}$.

$\mathbb{G}_1$: We guess parties P, Q output by the adversary in the last part of its game. In other words, the challenger must draw at random two values between 1 and $\mathsf{n}_{\mathcal{P}}$, such that those values correspond to those chosen by the adversaries. We lose a factor $\frac{1}{\mathsf{n}_{\mathcal{P}}^2}$.

$\mathbb{G}_2$: We now rule out that $H(\cdot, \mathsf{lkaux}) = (\cdot, \mathsf{lkaux}')$ for any $\mathsf{lkaux} \neq \mathsf{lkaux}'$. Trivially, if the converse were true,we could break the collision resistance of H with equal probability.

Note that now, since parties P and Q are registered on different platforms, since lkaux keys are unique, and since we have ruled out collisions, any honestly-generated attestations for P and Q will not link. The adversary's only hope is to forge an attestation for either one of those parties.

$\mathbb{G}_3$: At this point we rule out the fact that P's tuple $(\mathsf{AD}, \mathsf{linkedQuote}, \mathsf{AD}^*)$ was in fact part of a tuple $(P', \cdot, \mathsf{AD}, \mathsf{linkedQuote}, \mathsf{AD}^*) \in \mathcal{L}_{\mathsf{sign}}$ (with $P \neq P'$). If that were so, we could construct an adversary against the authenticated attestation scheme (since P purports to be P'). In so doing an important oracle will be the signature oracle oAuth added artificially in the authenticated attestation primitive; the latter will allow us to simulate $\mathsf{oLCAttest}$ queries. We lose $\varepsilon_{\text{Auth-attest}}$.

$\mathbb{G}_4$: We repeat the previous game hop for party Q, and lose $\varepsilon_{\text{Auth-attest}}$.

At this point, the adversary can no longer win the game.

4.5 Authorized Linked Attestation

So far, attestation has been viewed as a primitive, run by a single party (which can be of various types) and outputting an attestation. However, one of the most important requirements of attestation is that the actual quote only be given to authorized parties – which we call *attestation servers* [LK16].

We will define an *authorized linked attestation* protocol, which allows an attestation server to act as a verification party in the attestation procedures. The same server will also be the one to generate the auxiliary values required for the attestation (this provides freshness to the protocol). The server will also be responsible for linking multiple attestations.

4.5.1 Intuition

We provide a full formalization of authorized linked attestation below. However, we also believe it is useful to first give an intuitive understanding of what this primitive *is* and the security it wants to achieve.

In authorized linked attestation we consider a (single) attestation server S and platforms consisting of several types of components (as shown for linked attestation). The server will keep track of an evolving state, which is initially empty. However, as the server starts to attest various components, at every execution of the authorized attestation protocol, the server will output a verdict (indicating whether the component's individual attestation has failed or succeeded) and may – or may not – update its internal state. Intuitively, the state is meant to contain the *linking information* provided by each of the attesting components. After a number of attestations have been processed, the server might have enough information in its state to decide whether some of the components are linked or not.

The security notion we require for authorized linked attestation is threefold: (1) we require that parties only provide attestation guarantees to the actual attestation server; (2) we require that the contents of the attestation be actually indistinguishable from random for all unauthorized parties; (3) we require a similar kind of linking security as demanded in linked attestation see Section 4.4. However, as opposed to linked attestation, the adversary in this case can also play a Person-in-the-Middle role between honest components and the honest server, or it may attempt to replay messages or impersonate one or both parties. Finally, the adversary will be able to have oracle access to the secret key of any compromised component (this oracle access is parametrized in terms of a function space F of allowed functions).

4.5.2 Formalization

We will consider parties of multiple categories as for the linked-attestation primitive. We also define a special server entity, denoted S, which stores the following attributes:

- $(\mathsf{pk}, \mathsf{sk})$: a tuple consisting of a public key pk (assumed to be unique and known to all other parties including the adversary) and a private key sk known only to the server. We use $S.\mathsf{pk}$ to indicate the public key of party S, and $S.\mathsf{sk}$ to indicate its private key.
- $S.\mathsf{st}$: a value called *state*, which stores tuples of linked attestations which are susceptible to be linkable to each other.

We will consider an environment in which parties interact with each other in *sessions*. The session is run by two party *instances*, one of the attesting party and the other, of the attestation server. For a party P we denote by π_P^i the i-th instance of party P.

Just as in the case of linked attestation, parties store a set of keys $(\mathsf{pk}, \mathsf{sk})$, as well as a compromise bit γ.

Party instances use the same keys and have the same compromise bit as the party itself, but in addition keep track of the following session-specific attributes:

- sid: a session identifier, which will be useful in understanding which two party instances converse together.

- pidpk: the public key belonging to this instance's intended communication partner.
- T: a transcript of messages exchanged throughout a protocol session, in plaintext. Even if encryption is used at some point, parties append messages to their transcripts only after decrypting.
- α: this bit is originally set to 0, but can be changed to 1 if this party instance has accepted its partner as a legitimate entity to run the authorized linked attestation with.
- lst: this local state variable stores instance- (and protocol-) specific values, such as encryption keys, randomness, etc.

In addition server instances π_S^j keep track of the following attribute, which is the output of the immediate attestation process taking place:

- verdict: this attribute stores a bit, initially set to 0, which is flipped to 1 if the attestation server's instance has accepted the attestation received during that session.

We call two instances π_P^i and π_Q^j *partnered* if, and only if, the following conditions hold simultaneously: exactly one of P and Q is in fact the attestation server S; $\pi_P^i.\mathsf{pidpk} = Q.\mathsf{pk}$ and $\pi_Q^j.\mathsf{pidpk} = P.\mathsf{pk}$; and $\pi_P^i.\mathsf{sid} = \pi_Q^j.\mathsf{sid}$.

Authorized linking attestation is defined as the tuple of algorithms and protocols $\mathtt{ALA} = (\mathsf{ASetup}, \mathsf{AReg}, \mathsf{AAttest}, \mathtt{aALink})$ described as follows:

- $\mathsf{ASetup}(1^\lambda) \rightarrow (\mathsf{pparam}, S.\mathsf{pk}, S.\mathsf{sk}, S)$: on input the security parameter 1^λ (in unary), this algorithm outputs public parameters pparam, as well as the server handle S (such that S is equipped with newly generated keys pk, sk). The value pparam includes the maximal number of allowed disjoint linkable sets of parties, which we denote as L. The values pparam, S.pk, and S are public, S.sk remains private. The value S.pk is added as the first value in the set $\mathcal{PK}$.
- $\mathsf{AReg}(\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_L) \rightarrow \{(\mathsf{PK}_1, \mathsf{SK}_1), \ldots, (\mathsf{PK}_L, \mathsf{SK}_L)\}$: this algorithm keeps as state a number L of sets $\mathcal{S}_i$ originally set to $\emptyset$, and a vector of sets of public keys $\mathcal{PK}$ (also initialized to $\emptyset$). On input a number of subsets $\mathbf{s}_i$ ($i = 1, 2, \ldots, L$), this algorithm first checks that $\forall i, j, \mathbf{s}_i \cap \mathcal{S}_j = \emptyset$ (else the algorithm outputs $\perp$). If the relation is true, then the algorithm generates for each party $P_j \in \mathbf{s}_i$, (for all j, i) a tuple of public and private keys $P_j.\mathsf{pk}, P_j.\mathsf{sk}$, initializing P_j with those keys. We require the uniqueness of all the generated public keys. The subsets $\mathbf{s}_i$ are each added to greater sets $\mathcal{S}_i$. The algorithm groups the keys of all parties $P_j \in \mathbf{s}i$ in a pair of private/public key subsets: $(\mathsf{PK}_i, \mathsf{SK}_i)$, updating the i-th component $\mathcal{PK}[i]$ of $\mathcal{PK}$ as $\mathcal{PK}[i] \cup \mathsf{PK}_i$. All parties are given access to the public-key subsets (and more generally, to $\mathcal{PK}$).
- $\mathsf{AAttest}(\mathsf{pparam}, \mathcal{PK}, \pi_P^i, \pi_Q^j) \rightarrow (\mathsf{verdict}, S.\mathsf{st})$: this protocol is an interaction between two party oracles, such that exactly one of P, Q is S. The protocol yields a tuple of values verdict and S.st to the server (and no output for the other party). Both party oracles are assumed to update their attributes accordingly as the protocol unfolds.

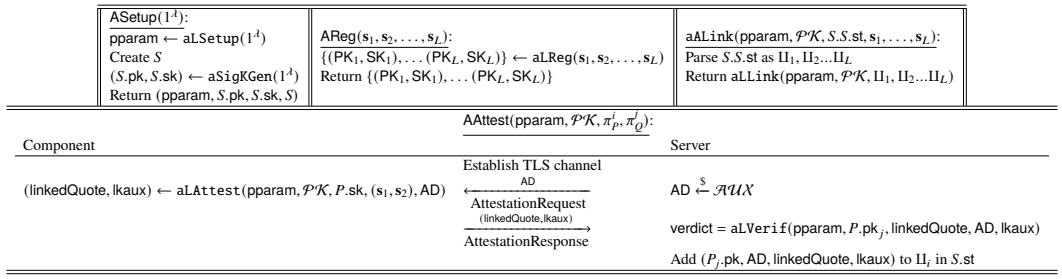


Figure 4.10 *Our authorized linked attestation scheme for 2 types of components.*

- aALink(pparam, $\mathcal{PK}, S.\mathsf{st}, \mathbf{s}_1, \ldots, \mathbf{s}_L$) : $0 \cup 1$: given the public parameters and public-key set, the server's current state, and a number of subsets of (purportedly-linked) parties, this algorithm outputs either 0 (the parties are not linked) or 1 (the parties are linked).

We require two types of *correctness* properties. First, we require that running the protocol between two honest parties yields a verdict of 1 (accept) on the side of the attestation server. Secondly, we require that components that are linked at registration will be viewed as linked by the aALink algorithm. More formally, we require that schemes ALA = (ASetup, AReg, AAttest, aALink) be such that:

- For all (pparam, S.pk, S.sk, S) $\leftarrow$ ASetup(1^λ) and for all parties $P \in \mathcal{S}_i$ for some $1 \leq i \leq L$, it holds that

$$(\mathsf{verdict}, \cdot) = \mathsf{AAttest}(\mathsf{pparam}, \mathcal{PK}, \pi_P^i, \pi_S^i)$$

  (any legitimate party will successfully attest to the legitimate server);

- For all (pparam, $\cdot, \cdot, S$) $\leftarrow$ ASetup(1^λ), for all subsets $\tilde{\mathbf{s}_1}, \tilde{\mathbf{s}_2}, \ldots, \tilde{\mathbf{s}_L}$ such that there exist sets $\mathbf{s}_i$ for $i = 1, 2, \ldots, L$ such that $\tilde{\mathbf{s}_i} \subset \mathbf{s}_i$ and AReg($\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_L$) was called and did not result in $\perp$, it holds that:

$$\mathsf{aALink}(\mathsf{pparam}, \mathcal{PK}, S.\mathsf{st}, \tilde{\mathbf{s}_1}, \ldots, \tilde{\mathbf{s}_L}) = 1$$

  (parties that are registered together can be linked through the server's state).

4.5.3 Construction

Our construction of the AuthAttprimitive can be seen in the Figure 4.10. We consider the existence on an underlying LinkedAtt scheme that we use for the aLSetup, aLReg and aLLink in a straightforward manner. However, the aLAttest algorithm is no longer a primitive, but a protocol between two instances of two parties, P and Q. For simplicity of exposition, we assume that the instance of Q is the server attesting the component identified by P.

The protocol proceeds as follows. First, P and Q execute the TLS protocol, with P playing the role of the client and Q playing the role of the server. The role of the TLS

protocol is two-fold: first, P authenticates the server, so that they can determine whether this party is allowed to obtain attestation data. Second, it leads to the establishment of a secure channel, such that the following messages can be passed on in a secure manner. Once the traffic key(s) established, the protocol continues as follows. First, the server uniformly randomly samples a nonce AD, which is embedded in the first message of the protocol, AttestationRequest. In response, the party P executes the `aLAttest` algorithm and the output, consisting of a $\mathsf{linkedQuote}$ and the linkage information lkaux, is then sent to the server. The server will subsequently update his state.

In order for two components to be linked by the server successfully, the following conditions have to be met. First, the two components' attestation must be valid (their associated verdicts equals 1). Second, the two lkaux must be subsets of each other; essentially, the key that the VM used as part of its attestation must be found in the lkaux provided by the hypervisor.

We note that if the server has at some point accepted the attestation of a component (thus updating its state to add the linking information), and if later a failed attestation occurs with respect to that component, the server updates state as follows: it ignores the linking information provided in the second attestation; and it removes prior linking information provided by that component.

4.5.4 Security

There are three fundamental properties we want `ALA` schemes to have: an authenticity guarantee for the attestation server (authorization); a confidentiality guarantee for the contents of the attestation (indistinguishability); and a linkability guarantee for honestly-behaving components (linking-security). The first notion, authorization, captures the fact that before reaching an accepting state, a (non-server) party must be sure that it is speaking to the legitimate server. The second notion, indistinguishability, essentially covers Person-in-the-Middle confidentiality for the attestation protocol. The last property, linking-security, refers to the fact that no PitM adversary with the ability to compromise components can convince an attestation server that a component is linked to another if that is not the case in reality. Although this last property might seem similar to the security notion for our linked attestation primitive, there is one important difference between the two: in linked attestation the adversary has access to essentially two ways to generate an attestation (depending on whether the component is honest or compromised), whereas in *authorized linked attestation* the adversary will have more leeway in combining attestation material across sessions. The stronger adversary in this section will thus make for a stronger primitive in the end.

The three security games we define are parametrized by a function space F and a security parameter λ. They start with the challenger running the setup algorithm and outputting pparam as well as the handle S and its public key $S.\mathsf{pk}$ to the adversary. Note that this will not give the adversary black-box access to S's attributes: it simply allows the adversary to later instantiate new attestation-protocol sessions for that server.

The adversary will then have access to some, or all of the following oracles:

- $\mathsf{oALAReg}(n_1, \ldots n_L) \rightarrow (\mathsf{PK}_1, \ldots \mathsf{PK}_L)$: the authorized linked user-registration

oracle creates a linked platform consisting of n_i components of the type indicated by $\mathcal{S}_i$. The challenger first instantiates subsets $\mathbf{s}_i$ as a tuple of n_i handles $P_{i,j}$, with $1 \leq j \leq n_i$ and then runs the algorithm $\mathsf{AReg}(\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_L)$, instantiating the parties with their keys and outputting the public key-sets to the adversary. The subset consisting of the list of subsets is added to $\mathcal{L}_{\mathsf{Reg}}$.

- $\mathsf{oNewSession}(P, Q) \rightarrow \pi_P^i$: on input the identity of a target party P and a partnering party Q, if both entities are correctly registered and exactly one of them is the server, then this oracle instantiates an instance of P whose partner will be instantiated as $\mathsf{pidpk}= Q.\mathsf{pk}$. Note that in order to observe an honest session between two parties, an adversary would have to create two partnered instances, one of P and the other of Q.

- $\mathsf{oSend}(M, \pi_P^i) \rightarrow M'$: this oracle simulates sending a message M to an instance π_P^i, and outputs the response M' of the party instance. If the input message takes a special value $M = \texttt{prompt}$ and P is the initiator of the protocol (*i.e.*, the first party to have to send a message), this will trigger π_P^i to output the first message in the protocol. We note that some messages, when sent, might trigger errors, leading to an output $M' = \perp$. Other messages might trigger the attributes of the party (or instance) to be modified.

- $\mathsf{oRevealState}(\pi_P^i) \rightarrow \mathsf{lst}$: on input a valid party instance, this oracle returns the value stored by the attribute lst of that instance.

- $\mathsf{oUseKey}(P, f, \mathsf{AD}) \rightarrow \mathsf{AD}'$: on input a compromised party P (not the server), a function $f \in F$, and an auxiliary input value AD, this oracle evaluates f on $P.\mathsf{sk}$ and AD.

- $\mathsf{oCompromise}(P) \rightarrow \mathsf{OK} \cup \perp$: on input a registered party $P \neq S$, this oracle turns the party's compromise bit to 1 and returns OK. If $P = S$ or the party has not been registered, the output is an error symbol $\perp$.

We now proceed to describe each of the three security experiments we consider for our authorized linked attestation primitive along with their corresponding theorem and proof.

4.5.4.1 The *authorization* game $\texttt{AuthSec}_{\lambda,\mathsf{F}}$

After the challenger runs the setup algorithm, the adversary $\mathcal{A}$ gets access to all the oracles described above. It ultimately stops with a `stop` message. We say $\mathcal{A}$ wins if, and only if, there exists an instance π_P^i such that $P \neq S$, for which the following conditions hold simultaneously:

- π_P^i ends in an accepting state, *i.e.*, $\pi_P^i.\alpha = 1$;

- There exists no server instance π_S^j such that π_S^j is partnered with π_P^i.

In other words, the adversary wins if it can make a registered party believe it has talked to the server when this is not the case.

Theorem 9 *Our construction is* $\texttt{AuthSec}_{\lambda,\mathsf{F}_{\mathsf{Sign}}}$ *secure if the TLS protocol provides server authentication.*

$$\Pr[\mathcal{A} \textit{ wins } \texttt{AuthSec}_{\lambda,\mathsf{F}}] \leq \varepsilon_{\textit{TLS-auth}}.$$

Proof. Note that in order to win this game, the adversary must make a party accept a session with the server, such that no matching server instance exists. This is against the server authentication property we assume of the TLS protocol.

4.5.4.2 The *linking* game $\texttt{AuthLink}_{\lambda,\mathsf{F}}$

After the challenger runs the setup algorithm, the adversary $\mathcal{A}$ gets access to all the oracles above. It ends by outputting a tuple $(P, \mathbf{s}_1, \ldots, \mathbf{s}_L)$: such that for all $1 \leq i \leq L$, $\mathbf{s}_i \subset \mathcal{S}_i$ and there exists a unique i^* such that $P \in \mathcal{S}_{i^*}$ and $\mathbf{s}_{i^*} = \emptyset$. The challenger sets $\tilde{\mathbf{s}_i} := \mathbf{s}_i$ for all $i \neq i^*$, and $\tilde{\mathbf{s}_i} := P$. Then the challenger evaluates: $b \leftarrow \texttt{aALink}(\mathsf{pparam}, \mathcal{PK}, S.\mathsf{st}, \tilde{\mathbf{s}_1}, \ldots, \tilde{\mathbf{s}_L})$. The adversary is said to *win* if, and only if the following conditions hold simultaneously:

- $b = 1$;
- There exists a party Q and an index $j^* \neq i^*$ such that $Q \in \tilde{\mathbf{s}_{j^*}}$ and P and Q were not output by the same $\mathsf{oALAReg}$ query.

In other words, for this second game, the adversary has to run several sessions between (potentially compromised) parties and the (honest) server, thus bringing the server's state to a point where linkage can be verified based on that state.

Theorem 10 *Our construction is* $\texttt{AuthLink}_{\lambda,\mathsf{F}_{\mathsf{Sign}}}$ *secure if the underlying primitive* $\texttt{LinkedAtt}$ *is* $\texttt{LinkSec}_{\lambda,\mathsf{F}_{\mathsf{Sign}}}$ *and TLS is at least (s)ACCE secure.*

Proof. A key observation for this game is that the adversary cannot impersonate a server or determine it to provide bad randomness. Instead, the adversary can compromise components and run TLS sessions on their behalf with the server, or try to obtain input from honest components instead. We distinguish between two types of adversary behaviours.

Say $\mathcal{A}$ has never queried $\mathsf{oCompromise}$ for some party P. If the adversary prompts P to run a session, then $\mathcal{A}$ will not actually know anything about the messages (so it cannot misbehave on the quote, the nonce, or anything else). If $\mathcal{A}$ runs the TLS session instead of P, it will learn the channel key, but will not be able to prompt P for the quote (since P wants to run TLS and not the attestation protocol, and since $\mathcal{A}$ cannot impersonate the server).

Say $\mathcal{A}$ queries $\mathsf{oCompromise}$ for some party P. Then the adversary can run TLS sessions on behalf of that party and query $\mathsf{oUseKey}$ in an attempt to get information on the quotes. However, in that case, the attestation of that component fails, except again if we break linked authentication security.

This essentially means that the adversary has no way to maul honestly-generated input to suit its purposes.

4.5.4.3 The *indistinguishability* game $\texttt{AuthInd}_{\lambda,\mathsf{F}}$

Once the challenger has finished the setup, it also draws a bit b at random. The adversary gets once more access to the oracles described above. It finally outputs a tuple (π_P^i, m_0, m_1), consisting of a party instance and two messages, such that: $|m_0| = |m_1|$ and $\pi_P^i.\alpha = 1$. The challenger uses its knowledge of π_P^i's state on input m_b (which is m_0 or m_1 depending on the challenger's hidden bit) to simulate outputting a message M_b which corresponds to the next protocol message of π_P^i as that party would have sent it. Clearly if the protocol requires messages be sent in plaintext, $M_b = m_b$. The instance π_P^i, as well as any of its partnering instances, are closed and may no longer be used in any oracle. The adversary may subsequently continue to use oracles at will (except on the instances closed above) and eventually outputs a guess $d \in \{0, 1\}$. We say the adversary *wins* if, and only if, the following conditions hold simultaneously:

- $d = b$;
- No oRevealState query was made for either π_P^i, nor for any instance π_S^j of the server such that π_P^i and π_S^j are partnered.

Theorem 11 *Our construction is* $\texttt{AuthInd}_{\lambda,\mathsf{F}_{\mathsf{Sign}}}$ *secure if the TLS channel provides (minimally) (s)ACCE security. Let* $q_{sessions}$ *be the number of sessions.*

$$\Pr[\mathcal{A} \textit{ wins } \texttt{AuthInd}_{\lambda,\mathsf{F}_{\mathsf{Sign}}}] \leq \frac{1}{q_{sessions}} \varepsilon_{TLS\text{-}sACCE}.$$

Proof. Like in the proof of authorization, the reduction here is immediate. The property of sACCE (which is already provided by TLS 1.2, whereas TLS 1.3 gives even stronger guarantees) implies that messages exchanged across the TLS channel are secure.

4.6 Implementation

We provide a proof-of-concept implementation of our authorized linked attestation scheme. The implementation consists of three parts, a client for the hypervisor, a client for the Virtual Machines, and an attestation server written in Python 3. We do not consider the underlying NFV or cloud infrastructure, since our scheme abstracts those environments and can be used in any kind deep-attestation scenario. Therefore, any computer equipped with a TPM 2.0 (which can also be emulated) and which has virtualization capacities suffices for the purposes of our implementation. We provide our code as well as a detailed tutorial on how to install and configure both the infrastructure [AFJ+].

The infrastructure. We summarize our testing architecture in Figure 4.11 (note that some of our tests use more than 2 VMs – up to 55).

Our *hypervisor* is a laptop running Ubuntu 20.04.3 (kernel version 5.11.0-40) with an Intel i7-10875H CPU, 32GB RAM and a STMicroelectronics TPM. We used KVM to turn this laptop into a hypervisor. For high attestation performance, we used full *virtual TPM implementation*, using QEMU [qem] with libtpms 0.7 [Ber] and swtpm 0.5 [SB].

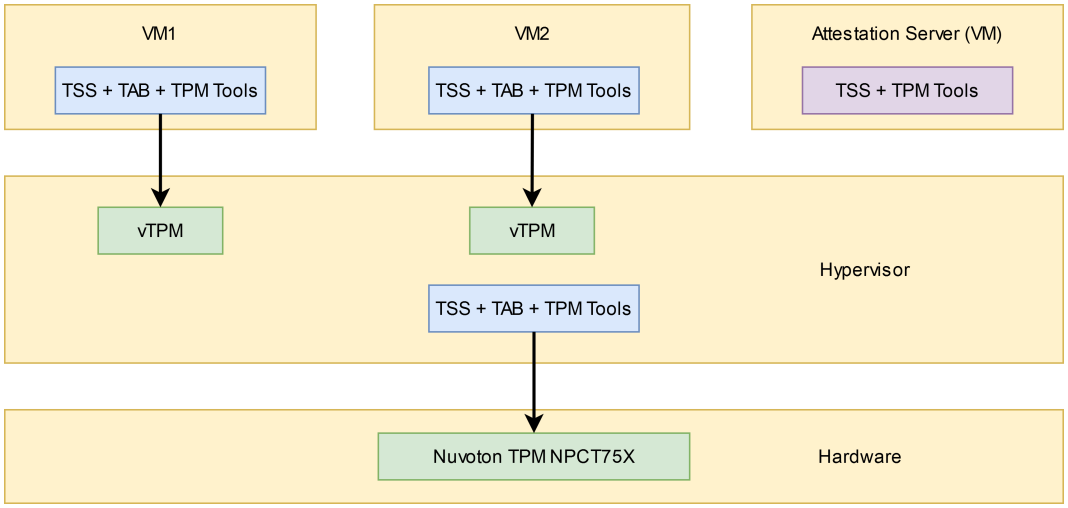


Figure 4.11 *Architecture for tests.*

All *virtual machines* are QEMU virtual machines (version 4.2.1) with 1 core and 512 RAM running Fedora 35 Cloud. The VM as well as the virtual TPM instances are managed using Vagrant and Vagrant-Libvirt plugin.

The hypervisor, server, and VMs communicate through a private network created with Vagrant. Thus, connection time is not considered in our tests.

To communicate with the TPM we used tpm2-tss, tpm2-abrmd and tpm2-tools from the tpm2-software [TPM]. Note that the tpm2-tss project implements the TPM software stack (TSS), which is an API specified by the Trusted Computing Group to interact with a TPM. The tpm2-abrmd implements the access broker and resources to manage concurrent access to the TPM and manage memory of the TPM by swapping in and out of the memory as needed (hardware TPM have limited memory).

The *attestation server* is also a virtual machine, with the same characteristics as those above. This allows us to test our implementation on a single machine. We establish a secure connection between the client and the server by using Python's SSL library.

Tests. We perform three types of experiments. The first is a comparison of hypervisor attestation time and VM attestation time. Although both those processes have some (very small) amount of noise, our values faithfully show the difference between attesting a component through the physical TPM – hypervisor attestation – and attesting it by using a virtual TPM – VM attestation.

We ran 100 attestations for the hypervisor and 100 attestations for a virtual machine. The results have high variance so Table 4.1 presents the minimum, the maximum, mean, and median value of those 100 trials. As expected, time for an attestation using a hardware TPM is much higher than using a vTPM.

As our second and third experiments we wanted to see how the overall runtime of our scheme evolves with the number of virtual machines that need to be attested, when the attestation is sequential or parallelized for the VM attestations. In both cases, each experiment first runs the attestation of the hypervisor, and then (sequentially or in parallel)

	min	median	mean	max
Hypervisor	3.22	5.30	5.68	11.55
VM	0.66	0.97	1.03	1.41

Table 4.1 *Minimum, median, mean and maximum time in second for attestation of a hypervisor and a virtual machine for 100 trials.*

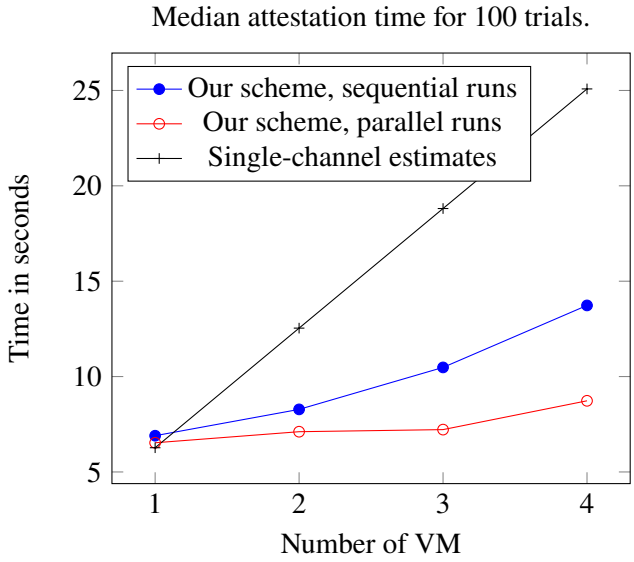


Figure 4.12 *Attestation time for 1, 2, 3 and 4 VMs. The upper curve is a single-channel attestation estimate, the middle curve is median runtime of our authorized linked attestation for sequential VM attestation, while the bottom curve shows the median runtime for our scheme with parallelization.*

the attestation of a varying number of VM (up to a maximum of 55). The results are plotted in Figure 4.12. We note that the runtime is not entirely linear. This is because in experiments 2 and 3 the initial attestation of the hypervisor (which only occurs once) takes larger time than the subsequent VM run-times.

Comparison to single-channel attestation. We did not implement single-channel attestation. However, since we have implemented hypervisor and VM attestations, we can theoretically estimate the run-time of single-channel attestation for a varying number of VMs – which we plot in Figure 4.12. Indeed, a single-channel attestation process for a single VM includes a VM attestation *and* a hypervisor attestation. If we want to run it for 2 VMs, then we need to perform 2 hypervisor attestations and 2 VM attestations. This cannot be easily parallelized either, because the same TPM has to run the attestations. This yields a much higher run-time, as depicted in Figure 4.12.

Comparison to multi-channel attestation. Although our method follows basic multi-channel attestation approaches, we do add an extra computation (a hash function computation) compared to traditional multi-channel attestation. In addition, we require a little extra memory overhead for both the attestation server and for each platform, so that the additional attestation keys are stored for each VM. There is also a slight transmission overhead, since

those keys are also sent upon attestation. However, the transmission overhead is negligible since it only appears for the hypervisor attestation (which occurs only once).

4.7 Conclusion

In this chapter, we described a means of guaranteeing layer-binding in deep-attestation without running into the complexity of single-channel attestation. Our construction achieves the best of both worlds, with a complexity similar to that of multi-channel attestation, but with the strong linkage properties provided in single-channel attestation. Our implementation results show that even for as few as 2 VMs, the time required for the (linked) attestation of the platform is halved with respect to single-channel attestation. We can do even better if the VM attestations are run in parallel. Our solution also enables easy linking, since the attestation server will only have to compare public keys, once the authentication and attestation of the components is successful.

In addition, we present a full, formal treatment of our new protocol, which we call *authorized linked attestation*. Our construction of authorized linked attestation is modular, building on primitives which have increasingly stronger properties. Our underlying assumption is a primitive called basic attestation. We show that in order to be able to prove security, we need that attestations be able to reflect (at least a statical) compromise of the component. In addition, we rely on a collision-resistant hash function, an EUF-CMA-secure signature scheme, and the the sACCE security of a TLS protocol (having AKE properties would be even better).

CONCLUSION

We first recall our contributions before presenting some perspectives that follow our works.

Contributions

We present several new cryptographic protocols proven in models we designed. Those protocols innovates in both asynchronous messaging applications and attestation for virtualized infrastructures.

The first contributions are related to *Post-Compromise Security*, a property guaranteeing to recover security after a compromise. Many protocols do not have this property and never heal, while some have this property with optimal healing (meaning that there is no better way to recover the security from a given compromise). Yet, there are in-between protocols where the healing is possible but may be enhanced. Considering the PCS as a spectrum rather than a binary notion leads to two main observations:

1. There are protocols where we can improve their PCS;
2. We should be able to measure the PCS.

Those two issues are the starting points of our first contributions.

We start from the description of the Signal protocol in order to point out avenues for improvement. From this analysis, we designed two protocols MARSHAL and SAMURAI, improving the PCS of Signal. Our approach is to stay as close as possible to the design of Signal allowing easy comparison between the protocols. We strenghen this conceptual contribution with an implementation showing that MARSHAL and SAMURAI remains practical even if there is an overhead.

The next step is to formalize the improvement made toward Signal and to check that the solutions we propose are indeed better, in terms of PCS, than the original protocol. Our goal not only consider those cases but a wide class of protocols ranging from Signal, SAMURAI, SAID (an other variant of Signal in the identity-based setting) and 5G procedures called handover. Our generic framework enables the comparison between those different protocols (and reinforce the improvement we have made with MARSHAL and SAMURAI toward the PCS of Signal).

In the last part of this thesis, we apply the methods of provable security to a different topic. The goal is to provide a provably secure scheme in the context of attestation schemes. Employing a scheme whose design is sound and provides provable security, is a good first step; however it is not enough to guarantee the security of the solution, once deployed in the real world. Attestation provides an additional guarantee, in ensuring that the components a protocol might be running on have not been modified. We therefore formalize and propose a provably-secure scheme for *Deep Attestation*. Our model, the first of this kind, allows us to show the security of a solution we designed in the computational model.

Perspectives

We can extend the work of the three main chapters of this thesis. First, our protocol SAMURAI could be developed into post-quantum paradigm or used for multi-party. Second, our model to quantify the PCS could be broaden to other protocols featuring additional properties. Finally, the work on deep attestation is also an interesting line of research.

Post-Quantum SAMURAI. Some works [BFG+22, BFG+20, HKKP21] appeared about key exchanged in the post-quantum paradigm that could directly help for the X3DH algorithm used by Signal. Industrials are also eager to propose Signal-like messaging protocol that could resist to post-quantum adversaries. A possible line of research is the design of SAMURAI in the post-quantum paradigm which could benefit optimal healing. The design of post-quantum protocols is critical since attackers could store communications and then, break the security with quantum computers.

Multi-Party SAMURAI. Signal is pairwise meaning that group communication does not scale efficiently. A future work to consider is the design of SAMURAI for n parties. This could yield optimal healing in group messaging. The issue for multi-party protocols is the key evolution. For two-party protocol, the sequence of messages is straightforward since when the speaker change, a new chain is created (*i.e.*, ping-pong style) so the key evolution is simple. An idea to solve this issue for multi-party would be to consider a n-dimension stage instead of a 2-dimension stage for a two-party (a stage is given by two coordinates for Signal). The user could choose which path to continue the communication to avoid conflicts.

Refining our metric. The framework we propose to analyse PCS does not feature all the protocols with the PCS property. We can add protocols like OTR [BGB04], Matrix [mat19] and Wire [Gmb21]. Moreover, the case of TLS 1.3 is interesting for our model and could lead to improvement toward PCS (particularly its session resumption feature).

Adding use cases is one step to consider but we can also refine the protocols we model. For instance, we do not consider out-of-order messages for Signal. This feature does not contradict the results from our metric but we simply omit it to facilitate the way we model protocols (including SAMURAI).

A metric for multi-user protocols. Our framework does not consider addition and revocation of members in a group. The work of [CHK21] analyses the effect of PCS on group messaging. Our approach, through our taxonomy, could reinforce their results.

Automated verification of our metric. Our choice of design for quantifying the PCS is the computational model. Yet, automated verification could also be possible (as long as global parameters can be managed) and we consider this line of research for future works. Also, the work of [KBB17] provides a methodology for automated verification of messaging protocols. It is inspirational since their approach could be used for SAMURAI.

Extending the model for deep attestation. Our model (and scheme) does not immediately account for other features of virtual infrastructures, such as privacy CAs, migrating VMs, multiple hypervisors managing the same VM, or even replacing TPMs. These aspects are left as future works.

Bibliography

[3GP20] 3GPP. System architecture for the 5G System (5GS). Technical Specification (TS) 23.501, 3rd Generation Partnership Project (3GPP), 10 2020. Version 16.0.0.

[3GP21] 3GPP. Procedures for the 5g system. Technical Specification (TS) 23.502, 3rd Generation Partnership Project (3GPP), 10 2021. Version 16.7.0.

[ABI+15] N. Asokan, Franz Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. SEDA: scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 964–975. ACM, 2015.

[ACD19] 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.

[AFJ+] Ghada Arfaoui, Pierre-Alain Fouque, Thibaut Jacques, Pascal Lafourcade, Adina Nedelcu, Cristina Onete, and Léo Robert. Implementation. `https://github.com/AnonymousDeepAttestation/deep-attestation`.

[AFJ+22] Ghada Arfaoui, Pierre-Alain Fouque, Thibaut Jacques, Pascal Lafourcade, Adina Nedelcu, Cristina Onete, and Léo Robert. A cryptographic view of deep-attestation, or how to do provably-secure layer-linking. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*, volume 13269 of *Lecture Notes in Computer Science*, pages 399–418. Springer, 2022.

[ASSW13] Frederik Armknecht, Ahmad-Reza Sadeghi, Steffen Schulz, and Christian Wachsmann. A security framework for the analysis and design of software attestation. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 1–12. ACM, 2013.

[BBB+19] 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.

[BBL+23] Olivier Blazy, Ioana Boureanu, Pascal Lafourcade, Cristina Onete, and Léo Robert. How fast do you heal? A taxonomy for post-compromise security in secure-channel establishment. *Usenix Security Symposium*, page to appear, 2023. https://eprint.iacr.org/2022/1090.

[BBR+22] 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.

[BCC04] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick D. McDaniel, editors, *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*, pages 132–145. ACM, 2004.

[BCK22] Chris Brzuska, Eric Cornelissen, and Konrad Kohbrok. Security analysis of the MLS key derivation. pages 2535–2553, 2022.

[BDF+18] 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.

[Ber] Stefan Berger. Library for TPM tools. https://github.com/stefanberger/libtpms.

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. volume 32, pages 586–615, 2003.

[BFG+20] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. Towards post-quantum security for signal's X3DH handshake. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*, volume 12804 of *Lecture Notes in Computer Science*, pages 404–430. Springer, 2020.

[BFG+22] Jacqueline Brendel, Rune Fiedler, Felix Günther, Christian Janson, and Douglas Stebila. Post-quantum asynchronous deniable key exchange and the signal handshake. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part II*, volume 13178 of *Lecture Notes in Computer Science*, pages 3–34. Springer, 2022.

[BFJ+22] Olivier Blazy, Pierre-Alain Fouque, Thibaut Jacques, Pascal Lafourcade, Cristina Onete, and Léo Robert. MARSHAL: messaging with asynchronous ratchets and signatures for faster healing. In Jiman Hong, Miroslav Bures, Juw Won Park, and Tomás Cerný, editors, *SAC '22: The 37th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, April 25 - 29, 2022*, pages 1666–1673. ACM, 2022.

[BGB04] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, WPES '04, page 77–84, New York, NY, USA, 2004. Association for Computing Machinery.

[BLOR21] Xavier Bultel, Pascal Lafourcade, Charles Olivier-Anclin, and Léo Robert. Generic construction for identity-based proxy blind signature. In Esma Aïmeur, Maryline Laurent, Reda Yaich, Benoît Dupont, and Joaquín García-Alfaro, editors, *Foundations and Practice of Security - 14th International Symposium, FPS 2021, Paris, France, December 7-10, 2021, Revised Selected Papers*, volume 13291 of *Lecture Notes in Computer Science*, pages 34–52. Springer, 2021.

[BNN09] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Security proofs for identity-based identification and signature schemes. *J. Cryptol.*, 22(1):1–61, 2009.

[BR93a] 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.

[BR93b] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.

[BR93c] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, pages 62–73. ACM, 1993.

[BSJ+17] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa*

Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III, volume 10403 of *Lecture Notes in Computer Science*, pages 619–650. Springer, 2017.

[BWS19] Robert Buhren, Christian Werling, and Jean-Pierre Seifert. Insecure until proven updated: Analyzing AMD sev's remote attestation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 1087–1099. ACM, 2019.

[CCG16] Katriel Cohn-Gordon, Cas 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. IEEE Computer Society, 2016.

[CCG+18] 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 David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1802–1819. ACM, 2018.

[CDDF20] Sébastien Campion, Julien Devigne, Céline Duguey, and Pierre-Alain Fouque. Multi-device for signal. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *Applied Cryptography and Network Security - 18th International Conference, ACNS 2020, Rome, Italy, October 19-22, 2020, Proceedings, Part II*, volume 12147 of *Lecture Notes in Computer Science*, pages 167–187. Springer, 2020.

[CDGM19] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. Seemless: Secure end-to-end encrypted messaging with less</> trust. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 1639–1656. ACM, 2019.

[CDV21] Andrea Caforio, F. Betül Durak, and Serge Vaudenay. Beyond security and efficiency: On-demand ratcheting with security awareness. In Juan A. Garay, editor, *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II*, volume 12711 of *Lecture Notes in Computer Science*, pages 649–677. Springer, 2021.

[CGCD+17] 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.

[CGL+11] George Coker, Joshua D. Guttman, Peter A. Loscocco, Amy L. Herzog, Jonathan K. Millen, Brian O'Hanlon, John D. Ramsdell, Ariel Segall, Justin Sheehy, and Brian T. Sniffen. Principles of remote attestation. *Int. J. Inf. Sec.*, 10(2):63–81, 2011.

[CHK21] Cas Cremers, Britta Hale, and Konrad Kohbrok. The complexities of healing in secure group messaging: Why cross-group effects matter. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1847–1864. USENIX Association, August 2021.

[CK01] 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.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002.

[CPZ20] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1445–1459. ACM, 2020.

[DDLR21] Jannik Dreier, Jean-Guillaume Dumas, Pascal Lafourcade, and Léo Robert. Optimal Threshold Padlock Systems. *Journal of Computer Security*, pages 1–34, 2021.

[DH21] Benjamin Dowling and Britta Hale. Secure messaging authentication against active man-in-the-middle attacks. In *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*, pages 54–70. IEEE, 2021.

[DLL12] 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.

[DSW09] Lucas Davi, Ahmad-Reza Sadeghi, and Marcel Winandy. Dynamic integrity measurement and attestation: towards defense against return-oriented programming attacks. In Shouhuai Xu, N. Asokan, Cristina Nita-Rotaru, and

Jean-Pierre Seifert, editors, *Proceedings of the 4th ACM Workshop on Scalable Trusted Computing, STC 2009, Chicago, Illinois, USA, November 13, 2009*, pages 49–54. ACM, 2009.

[DV19] F. Betül Durak and Serge Vaudenay. Bidirectional asynchronous ratcheted key agreement with linear complexity. In *Advances in Information and Computer Security - 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28-30, 2019, Proceedings*, pages 343–362, 2019.

[ETS17] ETSI. Network functions virtualisation(nfv); trust; report on attestation technologies and practices for secure deployments, 10 2017.

[FG94] 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.

[FG14] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of google's QUIC protocol. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1193–1204. ACM, 2014.

[FZL10] Xia Fei, Yanqin Zhu, and Xizhao Luo. Efficient identity-based signature scheme in the standard model. In *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, volume 5, pages V5–480–V5–483, 2010.

[GKB+19] Kartik Gopalan, Rohith Kugve, Hardik Bagdi, Yaohui Hu, Dan Williams, and Nilton Bila. Multi-hypervisor virtual machines: Enabling an ecosystem of hypervisor-level services. *Proceedings of the 2017 USENIX Annual Technical Conference, USENIX ATC 2017*, pages 235–249, 2019.

[Gmb21] Wire Swiss GmbH. Wire security whitepaper, 2021. `https://wire-docs.wire.com/download/Wire+Security+Whitepaper.pdf`.

[HEK+19] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 5greasoner: A property-directed security and privacy analysis framework for 5g cellular network protocol. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 669–684. ACM, 2019.

[HHWS18] Stefan Hristozov, Johann Heyszl, Steffen Wagner, and Georg Sigl. Practical runtime attestation for tiny iot devices. In *Proceedings of NDSS DISS*, 01 2018.

[HKKP21] Keitaro Hashimoto, Shuichi Katsumata, Kris Kwiatkowski, and Thomas Prest. An efficient and generic construction for signal's handshake (X3DH): post-

quantum, state leakage secure, and deniable. In Juan A. Garay, editor, *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II*, volume 12711 of *Lecture Notes in Computer Science*, pages 410–440. Springer, 2021.

[HKR15] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 15–44, 2015.

[JHJ05] Jun Jiang, Chen He, and Ling-ge Jiang. On the design of provably secure identity-based authentication and key exchange protocol for heterogeneous wireless access. In Xicheng Lu and Wei Zhao, editors, *Networking and Mobile Computing, Third International Conference, ICCNMC 2005, Zhangjiajie, China, August 2-4, 2005, Proceedings*, volume 3619 of *Lecture Notes in Computer Science*, pages 972–981. Springer, 2005.

[JLM+20] Matthieu Journault, Pascal Lafourcade, Malika More, Rémy Poulain, and Léo Robert. How to teach the undecidability of malware detection problem and halting problem. In Lynette Drevin, Suné von Solms, and Marianthi Theocharidou, editors, *Information Security Education. Information Security in Action - 13th IFIP WG 11.8 World Conference, WISE 13, Maribor, Slovenia, September 21-23, 2020, Proceedings*, volume 579 of *IFIP Advances in Information and Communication Technology*, pages 159–169. Springer, 2020.

[JMM19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 159–188. Springer, 2019.

[JS18] Joseph Jaeger and Igors Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 33–62. Springer, 2018.

[Kat10] Jonathan Katz. *Digital Signatures*. Springer, 2010.

[KBB17] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated Verification for Secure Messaging Protocols and Their Implementations: A

Symbolic and Computational Approach. In *2nd IEEE European Symposium on Security and Privacy* , pages 435 – 450, Paris, France, April 2017.

[KFZ+20] Boyu Kuang, Anmin Fu, Lu Zhou, Willy Susilo, and Yuqing Zhang. DO-RA: data-oriented runtime attestation for iot devices. *Comput. Secur.*, 97:101945, 2020.

[KM07] Neal Koblitz and Alfred J. Menezes. Another look at "provable security". *Journal of Cryptology*, 20(1):3–37, 2007.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6223 LNCS:631–648, 2010.

[LK16] Hagen Lauer and Nicolai Kuntze. Hypervisor-based attestation of virtual environments. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse, France, July 18-21, 2016*, pages 333–340. IEEE Computer Society, 2016.

[LMM+21] Pascal Lafourcade, Daiki Miyahara, Takaaki Mizuki, Léo Robert, Tatsuya Sasaki, and Hideaki Sone. How to construct physical zero-knowledge proofs for puzzles with a "single loop" condition. *Theor. Comput. Sci.*, 888:41–55, 2021.

[LRS20a] Pascal Lafourcade, Léo Robert, and Demba Sow. Fast short and fast linear cramer-shoup. In Gabriela Nicolescu, Assia Tria, José M. Fernandez, Jean-Yves Marion, and Joaquín García-Alfaro, editors, *Foundations and Practice of Security - 13th International Symposium, FPS 2020, Montreal, QC, Canada, December 1-3, 2020, Revised Selected Papers*, volume 12637 of *Lecture Notes in Computer Science*, pages 121–136. Springer, 2020.

[LRS20b] Pascal Lafourcade, Léo Robert, and Demba Sow. Linear generalized elgamal encryption scheme. In Pierangela Samarati, Sabrina De Capitani di Vimercati, Mohammad S. Obaidat, and Jalel Ben-Othman, editors, *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications, ICETE 2020 - Volume 2: SECRYPT, Lieusaint, Paris, France, July 8-10, 2020*, pages 372–379. ScitePress, 2020.

[LRS21] Pascal Lafourcade, Léo Robert, and Demba Sow. Fast cramer-shoup cryptosystem. In Sabrina De Capitani di Vimercati and Pierangela Samarati, editors, *Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021, July 6-8, 2021*, pages 766–771. SCITEPRESS, 2021.

[LWPM07] Peter A. Loscocco, Perry W. Wilson, J. Aaron Pendergrass, and C. Durward McDonell. Linux kernel integrity measurement using contextual inspection. In Peng Ning, Vijay Atluri, Shouhuai Xu, and Moti Yung, editors, *Proceedings of the 2nd ACM Workshop on Scalable Trusted Computing, STC 2007, Alexandria, VA, USA, November 2, 2007*, pages 21–29. ACM, 2007.

[mat19] An open network for secure, decentralized communication, 2019. `https://matrix.org/`.

[MP16a] M. Marlinspike and T. Perrin. The double ratchet algorithm, 2016. https://whispersystems.org/docs/specifications/doubleratchet/doubleratchet.pdf.

[MP16b] Moxie Marlinspike and Trevor Perrin. The x3dh key agreement protocol. *Signal*, page 11, 2016.

[MRL+20] Daiki Miyahara, Léo Robert, Pascal Lafourcade, So Takeshige, Takaaki Mizuki, Kazumasa Shinagawa, Atsuki Nagao, and Hideaki Sone. Card-Based ZKP Protocols for Takuzu and Juosan. In Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara, editors, *10th International Conference on Fun with Algorithms (FUN 2021)*, volume 157 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:21, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2002.

[OP01] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. *Lecture Notes in Computer Science*, 2001.

[PH10] 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.

[PR18] Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 2018.

[qem] The Qemu machine emulator. `https://www.qemu.org/`.

[RML+22] Léo Robert, Daiki Miyahara, Pascal Lafourcade, Luc Libralesso, and Takaaki Mizuki. Physical zero-knowledge proof and np-completeness proof of suguru puzzle. *Inf. Comput.*, 285(Part):104858, 2022.

[RMLM20] Léo Robert, Daiki Miyahara, Pascal Lafourcade, and Takaaki Mizuki. Physical zero-knowledge proof for suguru puzzle. In Stéphane Devismes and Neeraj Mittal, editors, *Stabilization, Safety, and Security of Distributed Systems - 22nd International Symposium, SSS 2020, Austin, TX, USA, November 18-21, 2020, Proceedings*, volume 12514 of *Lecture Notes in Computer Science*, pages 235–247. Springer, 2020.

[RMLM21] Léo Robert, Daiki Miyahara, Pascal Lafourcade, and Takaaki Mizuki. Interactive physical ZKP for connectivity: Applications to nurikabe and hitori. In Liesbeth De Mol, Andreas Weiermann, Florin Manea, and David Fernández-Duque, editors, *Connecting with Computability - 17th Conference on Computability in Europe, CiE 2021, Virtual Event, Ghent, July 5-9, 2021, Proceedings*, volume 12813 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2021.

[RMLM22] Léo Robert, Daiki Miyahara, Pascal Lafourcade, and Takaaki Mizuki. Card-based ZKP for connectivity: Applications to nurikabe, hitori, and heyawake. *New Gener. Comput.*, 40(1):149–171, 2022.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. *Proceedings of the ACM Conference on Computer and Communications Security*, (September):98–107, 2002.

[Rog16] Phillip Rogaway. The moral character of cryptographic work. Austin, TX, August 2016. USENIX Association.

[SB] David Safford and Stefan Berger. Software TPM emulator – swtpm. `https://github.com/stefanberger/swtpm`.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1984.

[Sho06] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *Cryptology ePrint Archive, Report 2004/332*, pages 1–33, 2006.

[TPM] TPM2_tools. `https://github.com/tpm2-software`.

[WBPE21] Jan Wichelmann, Sebastian Berndt, Claudius Pott, and Thomas Eisenbarth. Help, my signal has bad device! - breaking the signal messenger's post-compromise security through a malicious device. In Leyla Bilge, Lorenzo

Cavallaro, Giancarlo Pellegrino, and Nuno Neves, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment - 18th International Conference, DIMVA 2021, Virtual Event, July 14-16, 2021, Proceedings*, volume 12756 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2021.

Appendix A

Résumé Long

Pouvoir communiquer avec n'importe qui est devenu nécessaire dans notre société actuelle. Le besoin de sécurité à travers les outils permettant l'échange d'information est un besoin exprimé par de nombreux citoyens [1], notamment depuis les révélations d'Edward Snowden. Celui-ci a rendu publique une surveillance de masse opérée par des gouvernements et visant une large part de la population. Depuis ces révélations, de nombreux travaux (de recherches, mais aussi industriels) ont émergé pour assurer au mieux la sécurité des services de communication ainsi que de la vie privée des utilisateurs. L'un de ces services, appelé Signal, a connue un succès important lors de sa sortie notamment grâce à sa robustesse mais aussi pour sa facilité d'utilisation (d'autres outils existaient mais sans réussir à susciter l'intérêt des citoyens; par exemple PGP). La robustesse de Signal peut être exprimée à travers ses propriétés de sécurité:

- Asynchronicité : les communications entre participants peuvent se faire alors qu'un des deux participants ne soit en ligne. Ceci implique qu'Alice doit pouvoir envoyer un message à Bob à tout moment et Bob sera capable de lire ce message quand il reviendra en ligne;

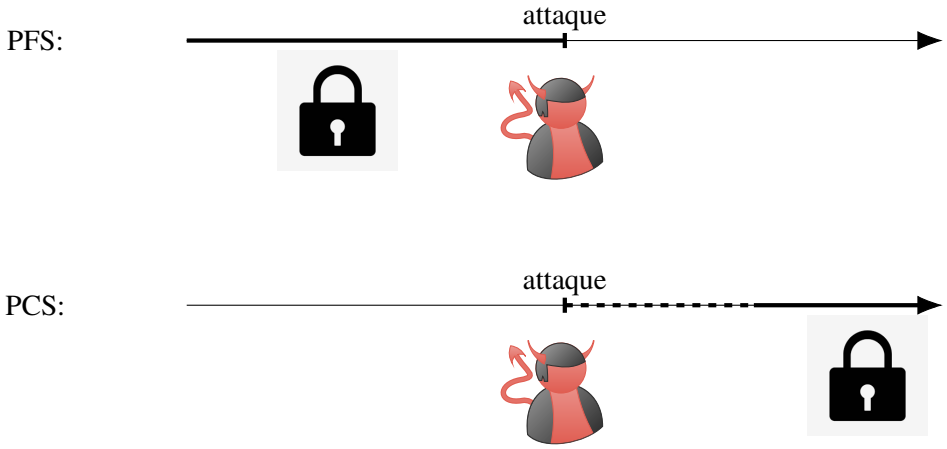
- Authenticité : les participants sont sûrs de parler à la bonne personne. Autrement dit, si Alice pense parler à Bob alors Bob est effectivement son partenaire de communication (et vice-versa);

- Sécurité Parfaite en Amont (*Perfect Forward Secrecy* – PFS) : si un adversaire réussit à compromettre à un moment donné la communication, alors tous les messages échangés *avant* l'attaque de l'adveraire restent sécurisés.

- Sécurité Après Compromission (*Post-Compromise Security* – PCS) : cette propriété est l'inverse de la PFS, autrement dit les messages échangés *après* l'attaque doivent pouvoir être à nouveau sécurisés, au bout d'un certain temps.

Ces deux dernières propriétés peuvent être illustrées de la manière suivante:

[1] Un exemple de ce changement de mentalité peut être illustré par un évènement survenu en 2021 avec WhatsApp; cette entreprise a vu un nombre important de désinscriptions de ses utilisateurs suite à une modification des conditions d'utilisation rendant un niveau de vie privée amoindri.



L'utilisation de Signal est prévue pour des communications à long terme; si Alice et Bob ont débuté une communication, elle doit pouvoir continuer sans avoir besoin de recalculer la mise en place d'une session. La propriété PCS devient donc cruciale dans ce contexte : si un attaquant réussit à compromettre un des participants (et donc à faire perdre la confidentialité de la communication) alors le PCS assure qu'au bout d'un moment, la communication "guérit" et annule tout dégât causé par l'attaque précédente. En pratique, il est préférable d'avoir une guérison rapide afin d'avoir le moins de messages possibles découvert par l'adversaire.

Les deux premiers axes de recherche présents dans cette thèse se focalisent sur cette notion de PCS. Dans un premier temps, nous proposons deux variantes de Signal qui améliorent la notion de PCS. Ensuite, nous proposons un modèle permettant d'évaluer ce temps de guérison quel que soit le protocole envisagé (possédant la notion de PCS). Le but est de pouvoir comparer des protocoles (qui a priori ne sont pas comparables) sous cette propriété de PCS afin de répérer le meilleur protocole (*i.e.*, celui avec la guérison la plus courte).

Dans ces deux axes de recherche, nous supposons que l'application Signal, installée dans un composant dédié (ordinateur, smartphone, tablette, etc) pour Alice et Bob, se comporte de la manière attendue. Cependant, notre description haut-niveau (dans le sens où nous avons abstraits des briques de base) ne prend pas en compte toute la chaîne de production pour arriver jusqu'à l'application qu'utilisent Alice et Bob. Une description correcte (avec des preuves de sécurité, calculatoires ou formelles) est indispensable [2] mais des failles peuvent survenir à chaque point intermédiaire jusqu'à l'utilisation finale de l'application. Par exemple :

- Le protocole doit être implémenté dans un langage de programmation. Il existe des bibliothèques cryptographiques pour que les briques de base soient bien utilisées mais certains choix de paramètres peuvent poser problèmes, le langage lui-même peut être source de failles.
- La partie logiciel de l'application vit dans un système (l'OS du composant *i.e.*, Windows, Linux, etc) comportant lui aussi des failles.

[2] Une illustration intéressante de ce fait est une remarque de Michel Raynal concernant les algorithmes (discutés de manière générale) : ce ne sont pas les algorithmes qui font voler un avion mais si les algorithmes utilisés ne sont pas corrects alors l'avion ne volera pas (ou pas longtemps en tout cas).

- Les composants physiques (la partie “hardware”) sont eux aussi sources de failles avec notamment les attaques par canaux auxiliaires. Ces attaques prennent en compte les fuites d'information par des phénomènes physiques (*i.e.*, la consommation d'énergie, la production d'ondes électro-magnétiques, les variations de température, etc).
- L'utilisation même du service par les utilisateurs. Par exemple, Alice pourrait laisser son smartphone ouvert, laissant la possibilité à un attaquant d'utiliser toutes sortes de moyens pour arriver à ses fins (simplement regarder, installer des applications, etc).

Il existe de nombreuses étapes, indépendantes des spécifications haut-niveau du protocole, pouvant amener à des attaques. Cependant, il existe aussi des solutions permettant d'avoir certaines assurances concernant la fiabilité des services voulus. L'une d'elles, appelée *attestation*, permet la verification d'une propriété pour une composante (réseau, machine virtuelle, IoT, etc). Les propriétés pouvant être évaluées sont nombreuses et dépendent de l'infrastructure analysée; ces propriétés peuvent être la géolocalisation, le contrôle d'accès à des ressources, l'intégrité de code.

Dans le dernier chapitre, nous nous concentrons sur une variante du processus d'attestation appelée *Deep Attestation*. Le but est de proposer le premier modèle de sécurité (pour de la sécurité prouvable) permettant d'évaluer la sécurité d'une solution que nous proposons pour le Deep Attestation. Cette solution, accompagnée d'une preuve de concept permettant d'évaluer sa faisabilité dans un contexte pratique, vise à améliorer l'équilibre entre la sécurité et la performance des solutions existantes (et standardisées).

A.1 Améliorer la PCS de Signal

La notion de PCS, introduite dans [CCG16], permet de garantir un niveau de sécurité après une compromission de clés. Suivant les clés révélées, un attaquant peut avoir accès à une partie de la communication jusqu'à un certain point où le protocole *guérit*. En ajoutant de l'aléa, l'attaquant n'a plus les informations nécessaires pour continuer à accéder à la communication et est éjecté du protocole. Cette propriété puissante, mais conditionnée par les clés révélées, a été analysée pour la première fois (dans le contexte du protocole Signal) dans [CGCD+17].

Le protocole Signal peut être rapidement décrit en 4 étapes:

- Un échange de clé intial : le protocole X3DH est utilisé avec l'aide d'un serveur;
- Un ratchet symétrique : dérivation d'une clé de message par une fonction de dérivation de clé (KDF) par un même participant sans attendre la réponse du destinataire; c'est cette étape qui assure la PFS.
- Un ratchet asymétrique : lorsque les rôles d'envoyeur/destinataire sont échangés, une nouvelle valeur Diffie-Hellman est insérée dans la dérivation de clé; c'est cette étape qui assure la PCS.
- Chiffrement authentifié : chaque message est chiffré avec une nouvelle clé (qu'on appelle *clé de message*); il faut associer à chaque envoi de message, des données auxiliaires via AEAD.

Nous nous intéressons à la transmission de messages, nous ne considérons donc pas l'échange de clé initial. Certains travaux comme [BFG+22, BFG+20, HKKP21] étudient en particulier cette étape de Signal. D'autres travaux se concentrent sur la structure de Signal pour des groupes [CPZ20] et multi-plateformes [CDDF20, WBPE21].

A.1.1 Contributions

Les garanties de PCS pour Signal sont limitées par deux facteurs :

- le manque d'authentification persistante : les clés sont modifiées mais cette évolution doit venir du bon participant.
- la fréquence d'utilisation du ratchet asymétrique : plus le ratchet asymétrique est utilisé souvent et plus le protocole guérit rapidement.

Notre but est de développer un protocole (en réalité deux) qui améliore la PCS tout en restant le plus proche possible de Signal. Nous restons proche de Signal en gardant la même structure, ce qui nous permet d'avoir une implémentation plus directe et aussi une comparaison claire de nos variantes. Nous proposons deux protocoles :

- MARSHAL [BFJ+22] (Messaging with Asynchronous Ratchets and Signatures for faster HeALing)

- et SAMURAI (Signal-like Asynchronous Messaging with Message-loss resilience, Ultimate healing and Robust against Active Impersonations)

Ces deux protocoles améliorent la PCS en garantissant une guérison plus rapide et une authentification persistante. Ces deux variantes ont le même niveau de sécurité mais SAMURAI gère mieux les valeurs stockées durant le protocole, ce qui rend ses performances mémoire plus pratiques que MARSHAL.

Ces propriétés ajoutées ont un coût. La principale raison de ce surcoût résulte du temps de calcul nécessaire pour calculer deux nouvelles opérations : une nouvelle valeur Diffie-Hellman et une signature pour chaque donnée auxiliaire supplémentaire.

Afin de faciliter la compréhension de nos contributions, nous donnons un exemple de conversation pour Signal. L'initiateur de la communication est Alice et le destinataire est Bob. Chaque message survient durant un *stage* qui est noté (x, y). La valeur y change lorsque l'envoyeur change ($y = 1$ correspond à Alice envoyant les messages; $y = 2$ correspond à Bob envoyant les messages et Alice les recevant). Chaque message est chiffré en utilisant une clé de message $\mathsf{mk}^{x,y}$ pour un stage (x, y).

Dans l'exemple ci-dessous, les notations [A] et [S] indiquent respectivement un ratchet asymétrique et symétrique. La sécurité (✓) et insécurité (×) des messages est donnée, en supposant qu'Alice s'est fait compromettre pour le message 2.

Envoyeur	**Key(s)**	**AD**	**Message**	MARSHAL/SAMURAI	**Signal**
Alice	$\mathsf{mk}^{1,1}$:	$(1, \mathsf{Rchpk}_A^1)$	Bonjour Bob	✓	✓
	[S] $\mathsf{mk}^{2,1}$	$(2, \mathsf{Rchpk}_A^1)$	ça va ?	×	×
	[S] $\mathsf{mk}^{3,1}$-$\mathsf{mk}^{17,1}$	$(3, \mathsf{Rchpk}_A^1)$-$(17, \mathsf{Rchpk}_A^1)$	(... *15 messages*)	✓	×
	[S] $\mathsf{mk}^{18,1}$	$(18, \mathsf{Rchpk}_A^1)$	Cinema ce soir ?	✓	×
Bob :	[A] $\mathsf{mk}^{1,2}$	$(1, \mathsf{Rchpk}_B^2)$	Salut Alice	✓	×
	[S] $\mathsf{mk}^{2,2}$	$(2, \mathsf{Rchpk}_B^2)$	Tout va bien, merci	✓	×
	[S] $\mathsf{mk}^{3,2}$-$\mathsf{mk}^{12,2}$	$(3, \mathsf{Rchpk}_B^2)$-$(12, \mathsf{Rchpk}_B^2)$	(... *10 messages*)	✓	×
Alice :	[A] $\mathsf{mk}^{1,3}$	$(1, \mathsf{Rchpk}_A^3)$	Parfait!	✓	✓

A.1.2 Description de Signal

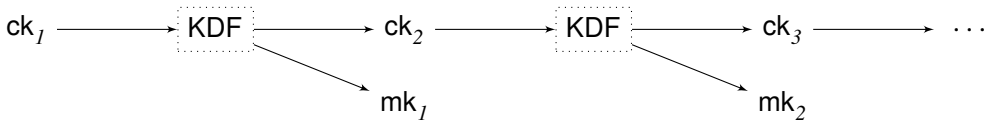
Avant de donner une description détaillée du protocole Signal, nous donnons une description de son mode opératoire générique. En effet, Signal est basé sur l'algorithme *double ratchet* proposé dans [MP16a].

Le double ratchet. Cet algorithme est divisé en deux sous-algorithmes, le *symétrique* et le *asymétrique*. L'idée est identique dans les deux cas : faire évoluer une clé à l'aide d'une fonction de dérivation de clé (KDF). Chaque sortie d'une KDF est considérée comme indistinguable d'une valeur aléatoire (ayant une distribution uniforme sur un espace donné; un exemple de KDF pourrait être une fonction pseudo-random PRF), et il doit être difficile

d'inverser la dérivation (*i.e.*, il n'existe pas d'algorithme en temps polynomiale qui puisse retrouver l'entrée avec seulement la sortie).

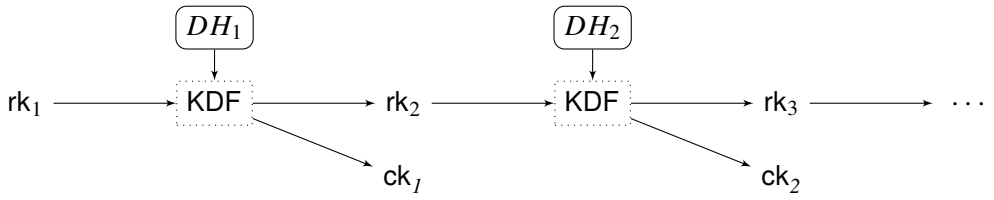
Le ratchet symétrique est déterministe dans le sens où en connaissant la première entrée, on est capable de retrouver toutes les autres sorties. Dans le cas de Signal, on appelle la séquence de clés produite une *chaîne*; l'entrée et une des sorties se nomment une clé de chaîne ck, l'autre sortie s'appelle la clé de message mk et c'est cette clé qui sert à chiffrer les messages.

Le ratchet symétrique s'illustre de la manière suivante :

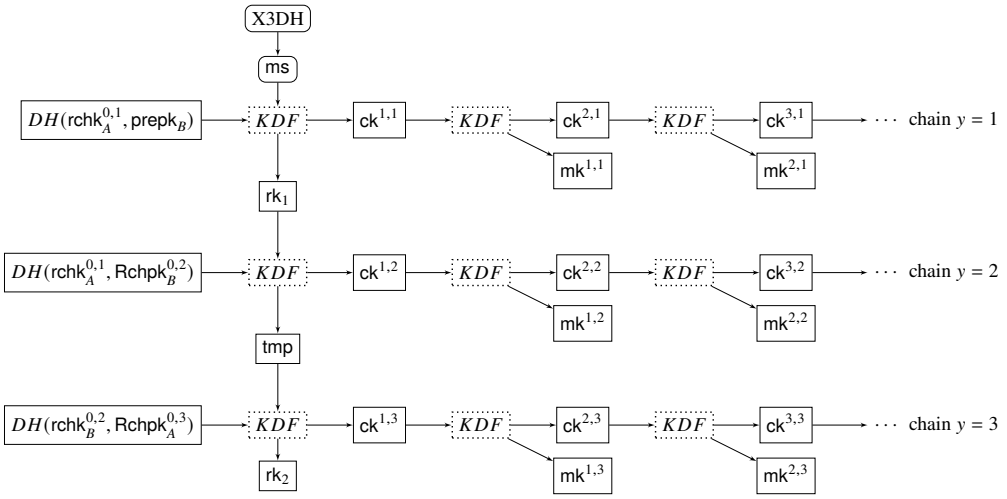


Le ratchet asymétrique fonctionne de la même manière sauf qu'à chaque appel de KDF, une valeur fraîche sera introduite. Dans le cas de Signal, cette valeur est une valeur Diffie-Hellman. Pour différencier les noms de clés utilisés, l'entrée et une des sorties de chaque KDF s'appellent une clé racine (root key) rk et l'une des sorties est la clé ck (utilisée dans le ratchet symétrique).

On peut représenter le ratchet asymétrique de la manière suivante :



Le protocole Signal. Ce protocole peut se décomposer en quatre étapes, l'enregistrement, la mise en place de la session, et l'envoi de messages (qui se décompose en deux parties, symétrique et asymétrique). Dans la figure suivante, nous donnons le planning de clés ainsi qu'un exemple de session entre Alice (l'initiatrice de la communication) et Bob (le receveur).



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

A.1.3 MARSHAL

Le protocole que nous proposons, Messaging with Asynchronous Ratchets and Signatures for faster HeALing (MARSHAL) se décompose comme Signal en quatre étapes : une phase d'enregistrement (pour chaque partie auprès d'un serveur), d'une phase de mise en place de la session et enfin la communication (composée de deux sous-parties). La nouveauté de MARSHAL est l'utilisation de clés de ratchet sur deux niveaux : des clés de ratchet propres à chaque utilisateur, et des clés de ratchet multi-utilisateurs. La figure suivante donne la description d'un exemple de session entre Alice et Bob pour les premiers *stages*. Les encadrés gris indiquent les modifications apportées par rapport à Signal; les données transmises sont différentes mais ne sont pas encadrées pour plus de clarté.

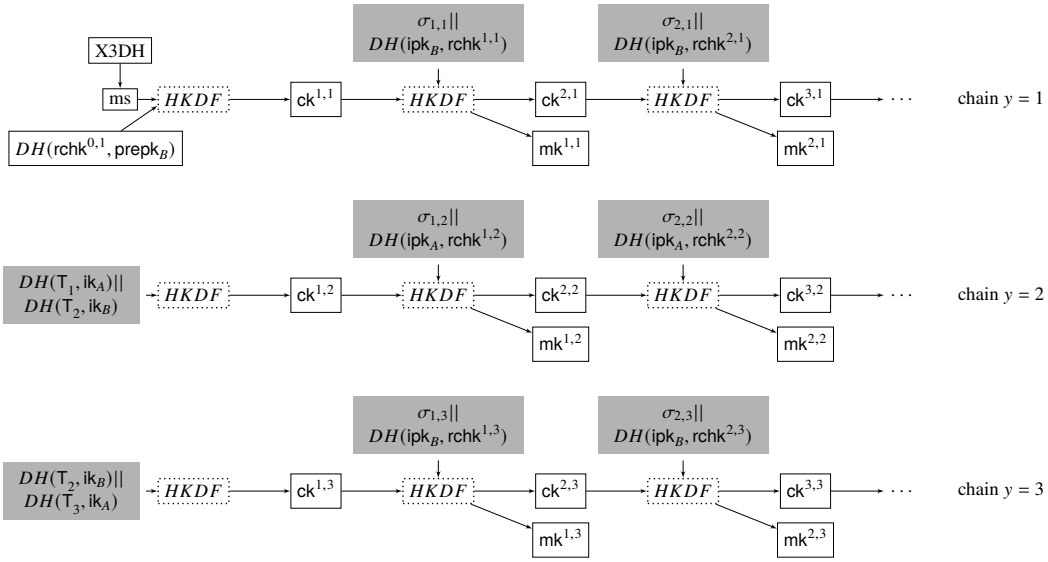
Alice ($\mathsf{ik}_A, \mathsf{ipk}_B, \mathsf{prepk}_B, \mathsf{ephpk}_B,$ T_0)		**Bob** ($\mathsf{ik}_B, \mathsf{ipk}_A, \mathsf{prek}_B, \mathsf{ephk}_B,$ T_0)
Session initialization: initiator Alice, responder Bob.		
$\mathsf{ek}_A, \mathsf{rchk}^{0,1},$ $\mathsf{t}_1, \mathsf{rchk}^{1,1} \xleftarrow{\$} \mathbb{Z}_q$; $\mathsf{T}_1 = g^{t_1}$; $\mathsf{Epk}_A = g^{\mathsf{ek}_A}$; $\mathsf{Rchpk}^{0,1} = g^{\mathsf{rchk}^{0,1}}$; $\mathsf{Rchpk}^{1,1} = g^{\mathsf{rchk}^{1,1}}$ $ms = \mathsf{prepk}_B^{\mathsf{ik}_A} \| \mathsf{ipk}_B^{\mathsf{ek}_A} \| \mathsf{prepk}_B^{\mathsf{ek}_A} \| \mathsf{ephpk}_B{}^{\mathsf{ek}_A}$ $\mathsf{ck}^{1,1} = \mathsf{HKDF}(\mathsf{prepk}_B^{\mathsf{rchk}^{0,1}} \| ms)$ $(\mathsf{ck}^{2,1}, \mathsf{mk}^{1,1}) = \mathsf{HKDF}(\mathsf{ck}^{1,1}, \sigma_{1,1} \| (\mathsf{ipk}_B)^{\mathsf{rchk}^{1,1}})$		
First message: stage $(1,1)$, Alice is the sender, Bob, the receiver.		
$AD_{y=1} = \mathsf{Epk}_A \| \mathsf{ipk}_A \| \mathsf{ipk}_B \| \mathsf{prepk}_B \| \mathsf{ephpk}_B \| \mathsf{T}_0 \| \mathsf{Rchpk}^{0,1} \| \mathsf{T}_1$ $AD_{1,1} = (1,1) \| \mathsf{Rchpk}^{1,1} \| \sigma_{1,1}$ $c_{1,1} = \mathtt{AEAD.Enc}_{\mathsf{mk}^{1,1}}(M_{1,1}; AD_1 \| AD_{1,1})$	$\xrightarrow{c_{1,1}, \mathtt{SIGN}_{sk_A}(c_{1,1}),\ pk_A, \mathtt{SIGN}_{\mathsf{ik}_A}(pk_A)}$	Verify signature on pk_A and $\sigma_{1,1}$ $ms = \mathsf{ipk}_A^{\mathsf{prek}_B} \| \mathsf{Epk}_A^{\mathsf{ik}_B} \| \mathsf{Epk}_A^{\mathsf{prek}_B} \| \mathsf{Epk}_A^{\mathsf{ephk}_B}$ $\mathsf{ck}^{1,1} = \mathsf{HKDF}((\mathsf{Rchpk}^{0,1})^{\mathsf{prek}_B} \| ms)$ $(\mathsf{ck}^{2,1}, \mathsf{mk}^{1,1}) = \mathsf{HKDF}(\mathsf{ck}^{1,1}, \sigma_{1,1} \| (\mathsf{Rchpk}^{1,1})^{\mathsf{ik}_B})$ $M_{1,1} = \mathtt{AEAD.Dec}_{\mathsf{mk}^{1,1}}(c_{1,1})$.
ℓ-th **message**: stage $(\ell, 1)$, Alice is the sender, Bob, the receiver.		
$\mathsf{rchk}^{\ell,1} \xleftarrow{\$} \mathbb{Z}_q$, set $\mathsf{Rchpk}^{\ell,1} = g^{\mathsf{rchk}^{\ell,1}}$ $(\mathsf{ck}^{\ell+1,1}, \mathsf{mk}^{\ell,1}) = \mathsf{HKDF}(\mathsf{ck}^{\ell,1}, \sigma_{\ell,1} \| \mathsf{ipk}_B^{\mathsf{rchk}^{\ell,1}})$ $AD_{\ell,1} = (\ell,1) \| \{\mathsf{Rchpk}^{x,1}\}_{1 \le x \le \ell} \| \sigma_{\ell,1}$ $c_{\ell,1} = \mathtt{AEAD.Enc}_{\mathsf{mk}^{\ell,1}}(M_{\ell,1}; AD_1 \| AD_{\ell,1})$	$\xrightarrow{c_{\ell,1}, \mathtt{SIGN}_{pk_A}(c_{\ell,1}),\ pk_A, \mathtt{SIGN}_{\mathsf{ik}_A}(pk_A)}$	Verify leftover signatures $(\mathsf{ck}^{\ell+1,1}, \mathsf{mk}^{\ell,1}) = \mathsf{HKDF}(\mathsf{ck}^{\ell,1}, \sigma_{\ell,1} \| (\mathsf{Rchpk}^{\ell,1})^{\mathsf{ik}_B})$ $M_{\ell,1} = \mathtt{AEAD.Dec}_{\mathsf{mk}^{\ell,1}}(c_{\ell,1})$.
Switching speakers: Bob comes online and begins a new ratcheting chain.		
		$\mathsf{t}_2, \mathsf{rchk}^{1,2} \xleftarrow{\$} \mathbb{Z}_q$; $\mathsf{T}_2 = g^{\mathsf{t}_2}$, $\mathsf{Rchpk}^{1,2} = g^{\mathsf{rchk}^{1,2}}$ $\mathsf{ck}^{1,2} = \mathsf{HKDF}(\mathsf{T}_1^{\mathsf{ik}_B} \| \mathsf{ipk}_A{}^{\mathsf{t}_2})$ $(\mathsf{ck}^{2,2}, \mathsf{mk}^{1,2}) = \mathsf{HKDF}(\mathsf{ck}^{1,2}, \sigma_{1,2} \| (\mathsf{ipk}_A)^{\mathsf{rchk}^{1,2}})$
Bob's message, stage $(1,2)$: Bob is the sender, Alice is the receiver.		
Verify signature on pk_B and $\sigma_{1,2}$ $\mathsf{ck}^{1,2} = \mathsf{HKDF}((\mathsf{ipk}_B)^{\mathsf{t}_1} \| (\mathsf{T}_2)^{\mathsf{ik}_A})$ $(\mathsf{ck}^{2,2}, \mathsf{mk}^{1,2}) = \mathsf{HKDF}(\mathsf{ck}^{1,2}, \sigma_{1,2} \| (\mathsf{Rchpk}^{1,2})^{\mathsf{ik}_A})$ $M_{1,2} = \mathtt{AEAD.Dec}_{\mathsf{mk}^{1,2}}(c_{1,2})$	$\xleftarrow{c_{1,2}, \mathtt{SIGN}_{pk_B}(c_{1,2}),\ pk_B, \mathtt{SIGN}_{\mathsf{ik}_B}(pk_B)}$	$AD_{y=2} = \mathsf{T}_2$ $AD_{1,2} = (1,2) \| \mathsf{Rchpk}^{1,2} \| \sigma_{1,2}$ $c_{1,2} = \mathtt{AEAD.Enc}_{\mathsf{mk}^{1,2}}(M_{1,2}; AD_2 \| AD_{1,2})$

Chaque message est envoyé chiffré de bout-en-bout, avec des données auxiliaires qui

servent à indiquer au receveur comment dériver les clés de messages calculées par l'envoyeur. Dans la figure suivante, nous donnons le planning de clés de MARSHAL avec les valeurs de signature suivantes:

$$\sigma_{x,y} := \begin{cases} \mathtt{SIGN}_{sk_A}\left(\mathsf{T}_{y-1}||\mathsf{Rchpk}^{x,y}\right), \text{ pour } y \text{ impair} \\ \mathtt{SIGN}_{sk_B}\left(\mathsf{T}_{y-1}||\mathsf{Rchpk}^{x,y}\right), \text{ pour } y \text{ pair} \end{cases}$$

De même que précédemment, les encadrés gris indiquent les modifications par rapport à Signal.



A.1.4 SAMURAI

Le deuxième protocole que nous proposons s'appelle SAMURAI pour Signal-like Asynchronous Messaging with Message-loss resilience, Ultimate healing and Robust against Active Impersonations. Ce protocole fonctionne comme MARSHAL, la différence majeure se situe au niveau des performances en terme de mémoire.

En effet, lors de l'envoi de messages, les données auxiliaires sont de taille constante (comparativement à une évolution linéaire pour MARSHAL). Le planning des clés n'est pas le même comme indiqué sur la figure ci-dessous, en prenant la définition suivante :

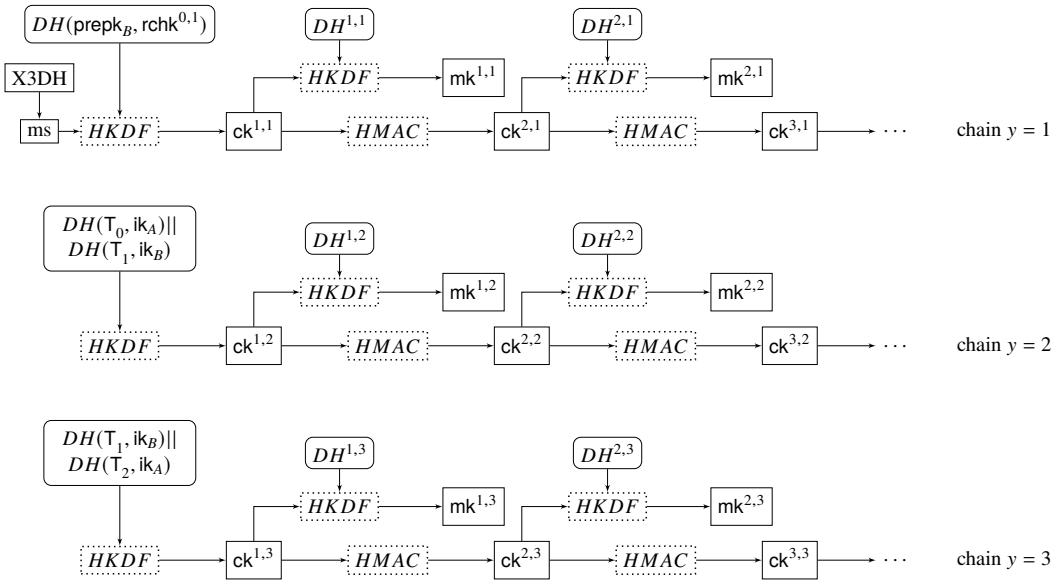
$$DH^{x,y} := \begin{cases} \sigma_{x,y}||DH(\mathsf{ipk}_B, \mathsf{rchk}^{x,y}), \text{ pour } y \text{ impair} \\ \sigma_{x,y}||DH(\mathsf{ipk}_A, \mathsf{rchk}^{x,y}), \text{ pour } y \text{ pair} \end{cases}$$

avec

$$\sigma_{x,y} := \begin{cases} \mathtt{SIGN}_{sk_A}\left(\mathsf{T}_{y-1}||\mathsf{Rchpk}^{x,y}\right), \text{ pour } y \text{ impair} \\ \\ \mathtt{SIGN}_{sk_B}\left(\mathsf{T}_{y-1}||\mathsf{Rchpk}^{x,y}\right), \text{ pour } y \text{ pair} \end{cases}$$

La valeur σ est une signature sur les éléments utilisés pour dériver la clé de message. C'est ce qui assure l'authentification persistante des valeurs prenant part à l'évolution des

clés. Le but des ces signatures est de minimiser la possiblité pour un attaquant de détourner la communication en injectant ses propres valeurs. En effet, la propriété PCS permet d'ajouter des valeurs aléatoires en cours de communication pour éjecter un attaquant passif mais un attaquant actif pourrait profiter de cette évolution de clé pour mettre ses propres valeurs et ainsi faire croire à un des partenaires de communication que la communication est toujours sécurisée (alors que l'adversaire a pris la place d'un des participants).



A.1.5 Conclusion

Nous proposons deux variantes d'un protocole de messagerie connu et largement plébiscité : Signal. Avec MARSHAL et SAMURAI, nous atteignons la meilleure sécurité possible en terme de PCS pour un coût en performance relativement faible. Contrairement à d'autres approches qui s'orientent vers des constructions génériques (notamment à base d'encapsulation de clé KEM), nous restons proches de la structure initiale de Signal.

Nos deux protocoles ont été construits après avoir constaté que le niveau amoindri de Signal pour le PCS était dû à deux facteurs :

- la fréquence de ratchet asymétrique, et
- l'absence d'authentification persistante.

La principale différence entre MARSHAL et SAMURAI se situe au niveau des performances (le niveau de sécurité étant le même). Nous voulions conserver une option de Signal appelée *out-of-order messages* (messages dans le désordre). Cette propriété permet de pouvoir retrouver un message même si les précédents ont été perdus (ou arrivent après). Pour MARSHAL, les données auxiliaires ont des tailles qui grossissent de manière linaire

en fonction du nombre de messages pour une chaîne donnée; alors que pour SAMURAI, la taille est constante.

A.2 Une métrique de guérison

De nombreux protocoles de messagerie ont vu le jour depuis ces dernières années. Chacun dispose de propriétés de sécurité spécifiques mais certaines sont reconnues comme étant nécessaires. Par exemple, la confidentialité des messages doit être assurée, mais aussi l'authenticité ou bien l'intégrité. Parmi ces propriétés se trouve la *Post-Compromise Security* (PCS) dont nous avons proposé une amélioration pour le protocole Signal lors du chapitre précédent. Le protocole Signal n'est cependant pas le seul à proposer la PCS; il y a aussi OTR [BGB04], Matrix [mat19], et Wire [Gmb21]. Une comparaison directe de ces protocoles en termes de PCS est impossible car chacun d'eux évolue dans un système donné. Par exemple, Signal possède un serveur semi-honnête (on lui fait seulement confiance dans le fait de stocker, sans modification, les clés d'utilisateurs) alors que SAID, une autre variante de Signal, basée sur l'identité, possède un centre de distribution de clés (KDC) qui connait tous les secrets associés aux utilisateurs.

A.2.1 Contributions

Dans ce chapitre, nous proposons un modèle permettant de quantifier le temps de guérison pour des protocoles possèdant la PCS. Pour définir formellement ce modèle, chaque protocole est modélisé en schéma SCEKE (Secure-Channel Establishment schemes with Key-Evolution, *i.e.*, Schémas avec Evolution de Clé pour un Etablissement de Canal Sécurisé). Le but d'un adversaire contre ce genre de protocole est d'apprendre le plus de messages possibles (qui sont chacun chiffrés avec une clé de message différente) après une attaque donnée. Toutes les attaques ne sont pas équivalentes, certaines peuvent donner lieu à des attaques actives, révéler plus ou moins de messages au cours de la communication. En plus de définir un modèle générique, nous proposons aussi une taxonomie exhaustive sur le type d'attaquant possible. Notre métrique mesure le nombre de messages "perdus" (*i.e.*, accessibles à l'adversaire) jusqu'au point où les messages sont à nouveaux hors d'atteinte de l'attaquant. Par exemple, la guérison optimale dans notre métrique vaut $(1, 0)$ alors que la pire vaut (∞, ∞) *i.e.*, le protocole ne guérit jamais. Ces cas sont évidents à traiter, mais notre métrique devient intéressante pour les cas se trouvant entre ces deux extrêmes.

Afin de montrer l'expressivité de notre modèle, nous comparons 4 schémas a priori très difficiles à comparer autrement :

- Signal, basé sur une infrastructure à clé publique;
- SAID, une variante de Signal basée sur l'identité;
- SAMURAI, notre variante de Signal décrite au chapitre précédent et spécifiquement construite autour de la PCS;

- enfin, des procédures de relais (*handover*) pour le réseau 5G. Ces procédures n'ont encore jamais été étudiées pour la PCS, c'est donc la première fois qu'une telle analyse est faite. De plus, nous proposons une variante visant l'amélioration de la PCS.

A.2.2 Notre métrique pour les SCEKE

Nous formalisons les protocoles SCEKE avant de pouvoir utiliser ce modèle sur des protocoles existants.

Nous considérons les protocoles avec deux types de participants :

- les utilisateurs P appartenant à un ensemble $\mathcal{P}$;
- un super-utilisateur $\hat{S}$, qui est une entité à part entière (*e.g.*, le serveur dans Signal).

Une fois que les utilisateurs se sont enregistrés auprès du super-utilisateur, ils peuvent commencer des sessions entre eux. Après un échange de clé standard, une session de protocole se déroule entre deux instances de participants. La i-th instance de P est notée π_P^i.

Du point de vue des utilisateurs, chaque session a trois phases :

- une initialisation : elle ne survient qu'une seule fois par instance (au début);
- une phase d'envoi de messages : elle intervient lorsqu'une instance envoie des messages;
- une phase de réception de messages : elle intervient lorsqu'une instance reçoit des messages.

Les deux dernières phases sont cruciales dans notre modèle, ce sont elles qui rendent compte de notre métrique car elles représentent la notion de stage. Un stage correspond à l'ensemble des opérations et valeurs permettant l'établissement d'une clé de message.

Le modèle d'adversaire. L'adversaire que nous considérons est un algorithme probabiliste calculant en temps polynomial. Il manipule les participants honnêtes via des oracles (algorithmes retournant une valeur définie mais dont le fonctionnement n'est pas explicite). Il a un contrôle total sur le réseau dans lequel évolue le protocole. Nous présentons les types d'adversaire en fonction de ses capacités.

Un adversaire possède une certaine *portée*, cela signifie qu'il peut révéler des clés (mais pas forcément toutes). Nous divisons la portée d'un adversaire en trois parties :

- locale : l'adversaire ne peut révéler que des clés qui ne sont utilisées qu'une seule fois dans la session;
- moyenne : l'adversaire peut, en plus du cas précédent, révéler des clés apparaissant plusieurs fois dans une même session (mais pas dans d'autres sessions);
- globale : l'adversaire peut révéler toutes les clés.

En plus de la portée, nous définissons la *puissance* de l'adversaire. Celle-ci est divisée en deux parties :

- active : l'adversaire peut directement interférer avec le fonctionnement du protocole. Il peut stoper, relancer, modifier des messages;

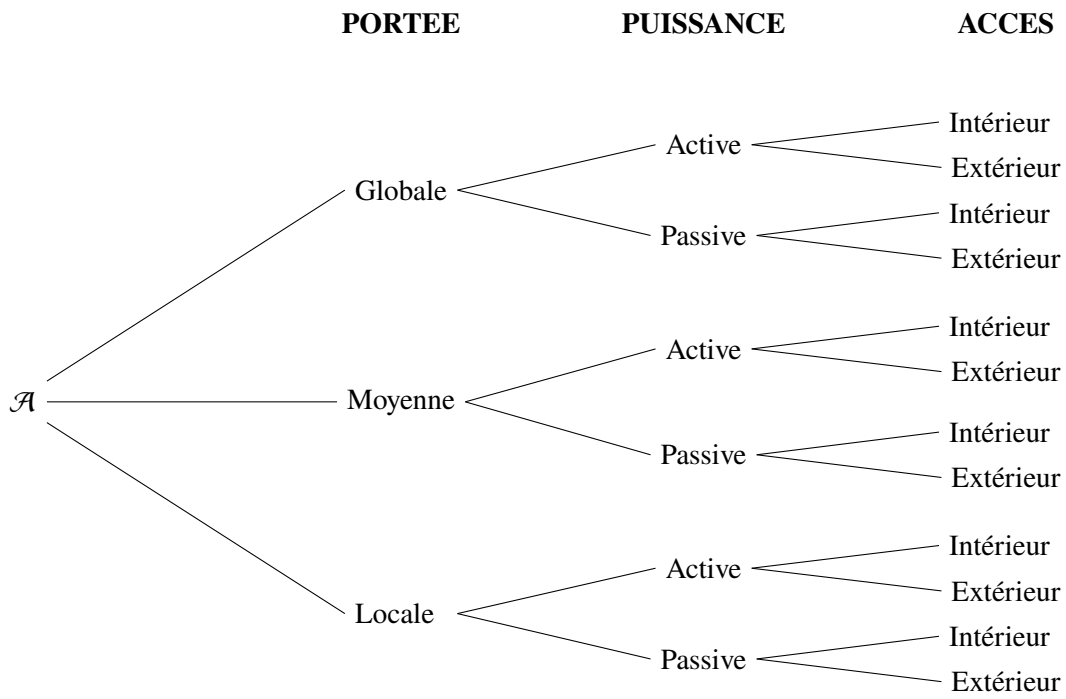
- passive : une fois l'attaque effectuée, l'adversaire ne peut plus modifier le fonctionnement du protocole.

Enfin, le dernier critère que nous considérons est celui d'*accès*. Ce critère permet d'analyser les protocoles centralisés et les protocoles décentralisés tout en permettant leur comparaison :

- intérieur : cet adversaire est le super-utilisateur $\hat{S}$.

- extérieur : cet adversaire ne reçoit pas les données de $\hat{S}$, et ne peut pas le corrompre au cours de la session.

On peut résumer la classification d'adversaire selon la figure suivante :



A.2.3 Résultats

Nous donnons l'ensemble des résultats dans le tableau suivant :

Extérieur	Portée	Signal	SAMURAI	SAID	*5G-SCEKE*	*5G-SCEKE*$^+$
Passive	Globale	$(\infty, 2)$	$(1, 0)$	$(\infty, 2)$	(∞, ∞)	$(1, 0)$
	Moyenne	$(\infty, 2)$	$(1, 0)$	$(\infty, 2)$	(∞, ∞)	$(1, 0)$
	Locale	$(\infty, 1)$	$(1, 0)$	$(1, 0)$	$(\infty, 1)$	$(1, 0)$
Active	Globale	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)
	Moyenne	(∞, ∞)	$(1, 0)$	(∞, ∞)	(∞, ∞)	(∞, ∞)
	Locale	$(\infty, 1)$	$(1, 0)$	$(1, 0)$	$(\infty, 1)$	$(1, 0)$

Intérieur	Portée	Signal	SAMURAI	SAID	*5G-SCEKE*	*5G-SCEKE*$^+$
Passive	Globale	$(\infty, 2)$	$(1, 0)$	$(\infty, 2)$	(∞, ∞)	(∞, ∞)
	Moyenne	$(\infty, 2)$	$(1, 0)$	$(\infty, 2)$	(∞, ∞)	(∞, ∞)
	Locale	$(\infty, 1)$	$(1, 0)$	$(\infty, 1)$	(∞, ∞)	(∞, ∞)
Active	Globale	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)
	Moyenne	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)
	Locale	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)

Nos résultats montrent que les protocoles SAMURAI, SAID et notre amélioration (*5G-SCEKE*$^+$) pour la procédure de relais 5G possède le meilleur taux de guérison contre des adversaires locaux, passifs et extérieurs. Les bons résultats de SAMURAI et *5G-SCEKE*$^+$ ne sont pas dûs aux même raisons. SAMURAI utilise plus fréquemment un ratchet asymétrique (qui implique donc plus de calculs) et une authentification persistante (qui a pour conséquence directe d'enlever toute forme de déniabilité) alors que *5G-SCEKE*$^+$ insère des valeurs aléatoires au cours d'une session grâce à la présence d'un deuxième canal sécurisé. Enfin, il est important de noter que le cas d'adversaire actif intérieur est difficile à gérer mais reste un but à rechercher. Nos résultats montrent clairement cette difficulté à gérer les attaques du milieu.

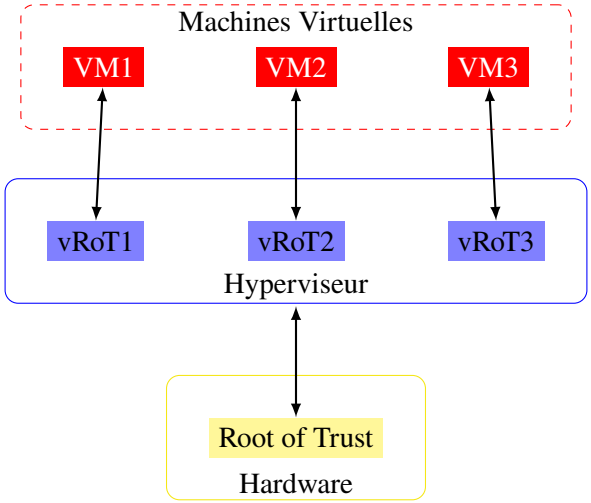
Nos résultats, bien que pouvant éclaircir des problèmes encore obscurs, possèdent aussi des désavantages. En effet, nous ne modélisons que des protocoles à deux participants, excluant donc des protocoles multi-utilisateurs comme ART ou MLS. Cependant, il reste encore d'autres protocoles pouvant être modélisés avec notre approche comme TLS 1.3, qui est pour le moment laissé en travaux futurs.

A.3 Attestation en profondeur

Notre monde est de plus en plus interconnecté, avec un besoin de réseaux flexibles et dynamiques toujours plus grandissants. Dans un environnement comme le cloud ou le réseau 5G, la technologie de virtualisation permet une mise à l'echelle facile et spécifique à la demande des utilisateurs. En effet, les machines virtuelles peuvent être facilement mises en place, mises à l'échelle ou bien déplacées sur n'importe quel type d'infrastructure physique permettant ainsi de meilleures performances et une plus grande sécurité. Cependant, de nouveaux besoins ont vu le jour; par exemple dans le domaine de la santé numérique (e-santé) où la géolocalisation est primordiale. Un outil permet de vérifier si ces infrastructures ont un état de fonctionnement valide, c'est *l'attestation*.

Le processus d'attestation permet à une entité indépendante de la platerforme de vérifier si cette plateforme a un comportement attendu, si son état n'a pas été modifié de manière

inattendue. Le but est de vérifier que certaines propriétés de sécurité sont vérifiées. Par exemple, un serveur pourrait vérifier que le code source d'une machine virtuelle est bien valide (*i.e.*, aucun code malicieux n'a été inséré). Nous nous intéressons à ce cas, où des machines virtuelles sont gérées par un hyperviseur qui lui-même interagit avec un composant physique nommé "racine de confiance" (*root of trust*). On peut représenter cette structure de la manière suivante :

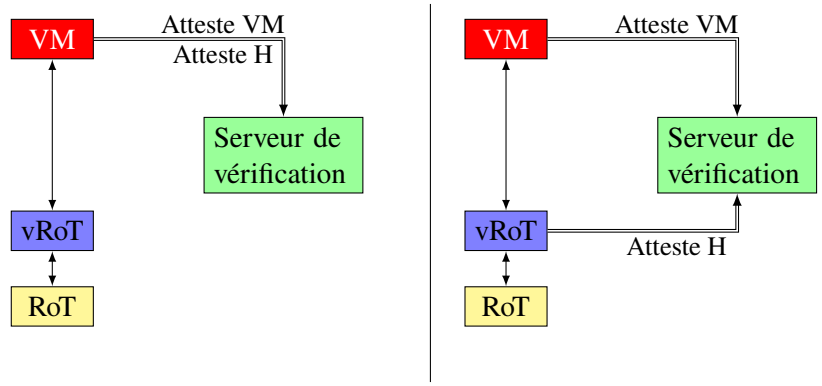


Il existe actuellement deux solutions pour l'attestation en profondeur :

- L'attestation via un seul canal : chaque VM est attestée en même temps que l'hyperviseur qui la gère. Cette solution est sûre dans le sens où une VM est liée à son hyperviseur mais la mise à l'échelle est très coûteuse, l'hyperviseur est attesté à chaque fois qu'une VM l'est.

- L'attestation par canal multiple : cette solution atteste l'hyperviseur et les machines virtuelles indépendamment. Cette fois la mise à l'échelle est efficace (une seule attestation pour l'hyperviseur) mais la sécurité n'est plus garantie, les VM ne sont plus liées à l'hyperviseur.

Nous représentons ces solutions avec le schéma ci-dessous.

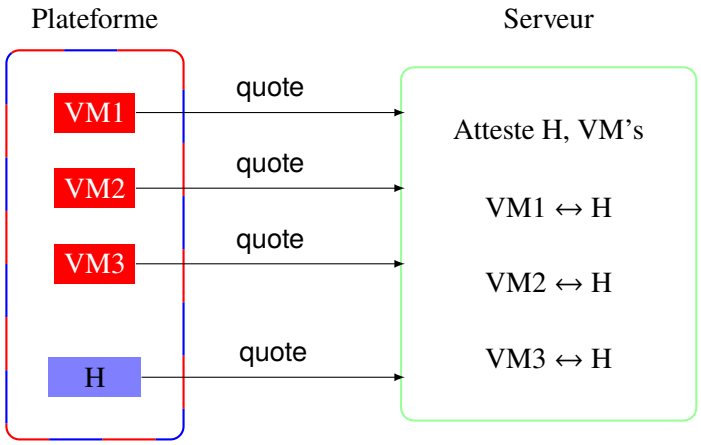


Securisé **Efficace**

A.3.1 Contributions

Nous prenons le meilleur des deux solutions standardisées (unique/multiple canal) pour l'attestation en profondeur pour obtenir un outil liant des machines virtuelles et des hyperviseurs avec une efficacité raisonnable (dans le sens où les performances se rapprochent de celles des deux solutions). Notre solution est simple, mais élégante, et utilisant des standards cryptographiques pour assurer qu'une attestation d'un hyperviseur est bien liée aux machines virtuelles dont il a la charge. Les contributions sont sur 3 points:

1. **Un schéma cryptographique :** Notre schéma assure la sécurité et l'efficacité pour l'attestation en profondeur. Pour cela, chaque hyperviseur et chaque machine virtuelle ne s'atteste qu'une seule fois. Chaque VM possède une paire de clés dont la partie publique fait partie de l'attestation même de l'hyperviseur qui la gère (qui lui-même est attesté par la racine de confiance physique). Afin d'authentifier ces clés, elles sont incluses dans un nonce, et transmises par le serveur de vérification. Si l'attestation de l'hyperviseur réussit, alors le serveur de vérification peut lier cet hyperviseur avec des VM qui s'attesteraient ultérieurement. Si l'attestation de l'hyperviseur échoue, alors l'ensemble des clés publiques ne peuvent pas être de confiance. Notre solution peut s'illustrer de la manière suivante:



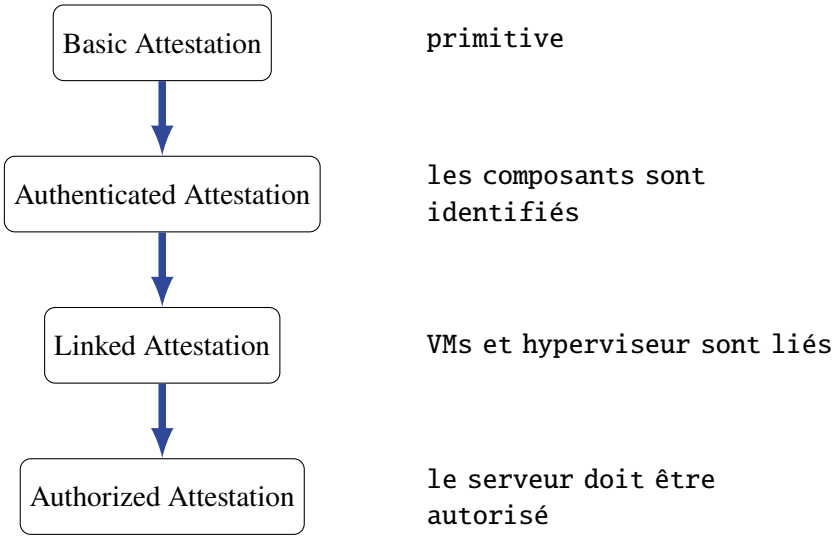
2. **Prouver la sécurité d'une attestation liante en étant autorisé** Un avantage clair de notre approche est d'avoir complètement formalisé et prouvé les garanties de sécurité. Nous utilisons une approche par série de jeux, permettant la construction de primitives de plus en plus fortes basées sur des plus faibles. Notre but est d'obtenir une primitive d'attestation liante et autorisante (ALA) : chaque composant s'atteste individuellement à une entité autorisée à vérifier pour avoir leurs attestations liées. Nous réglons ainsi le problème où des VM pourraient ne pas appartenir à un hyperviseur car chaque attestation de VM est maintenant liée à un hyperviseur (et casser cette hypothèse reviendrait à résoudre un problème reconnu difficile).

   ALA a trois propriétés :

   - l'autorisation : seulement un serveur dédié peut effectuer une attestation;

- l'indistinguabilité : aucun attaquant au milieu ne pourrait deviner qu'un seul bit d'information durant la communication entre des participants honnêtes;
- le liage : un serveur peut détecter si deux composants sont liés ou pas.

Nous formalisons une série de primitives dont la dernière est un échange de clé authentifiée. Chaque primitive est d'interêt indépendant et possède une sécurité de plus en plus forte. Cette approche a deux avantages : elle permet d'utiliser des primitives plus faibles en boîte noire pour former des primitives plus fortes; et elle permet aussi d'avoir des preuves plus directes et plus simples. Nous pouvons résumer notre approche de la manière suivante:



Nous commençons notre construction par un schéma basique d'attestation retournant simplement un résultat binaire (oui/non). Nous supposons donc une sécurité par définition. Sa fonctionnalité est simple : si le résultat de l'attestation retourne "non" alors le composant est malicieux mais retournera "oui" si le composant est honnête. Basés sur cette hypothèse, nous construisons une série de mécanismes cryptographiques pour ajouter des propriétés de sécurité afin de resister à des adversaires ayant plus de pouvoir. La première étape est de rajouter de l'authentification, ce qui assure qu'un composant pourra toujours s'attester avant une corruption mais pas après. Ensuite nous ajoutons la propriété de liage qui permet de lier certains composants entre eux (les VM avec l'hyperviseur qui les gère).

3. **Implémentation :** Nous avons effectué une preuve de concept pour assurer que notre solution a bien les performances attendues, et est capable d'être appliquée dans un contexte pratique. L'architecture est composée d'un hyperviseur et de plusieurs VM (jusqu'à 55). Cela montre que notre solution est plus performante que la solution à canal unique et ajoute un petit surplus (un calcul de fonction de hashage) comparé à la solution de canal multiple.

Ce travail est le premier à formaliser, en utilisant des outils cryptographiques, l'attestation en profondeur. Ce traitement est plus difficile que la conception en soi de la solution car

l'attestation est un processus générique présentant de nombreuses classes d'algorithmes qui ont chacun des buts différents. Ainsi, nous ne faisons qu'effleurer le problème mais espérons – à part pour le côté pratique de nos constructions – que le traitement cryptographique, primitives, et preuves sont indépendamment intéressantes pour cet axe de recherche.

A.4 Conclusion

Nous présentons dans cette thèse plusieurs protocoles cryptographiques prouvés dans des modèles qui font, eux aussi, partie de nos contributions. Ces protocoles innovent dans le domaine des applications de messagerie asynchrone ainsi que pour les infrastructures virtualisées.

Les premières contributions concernent la notion de Sécurité Après-Compromission (PCS), une propriété garantissant de retrouver une sécurité après une attaque. Beaucoup de protocoles n'ont pas cette propriété et ne guérissent jamais, alors que d'autres ont cette propriété et guérissent de manière optimale (autrement dit, il n'y a pas de meilleure guérison possible). Il existe aussi des protocoles ayant la PCS mais celle-ci peut être améliorée. La PCS est donc une notion variant d'un extrême à un autre, pouvant prendre des valeurs intermédiaires. Cette observation nous mène à considérer ces deux points:

1. Il existe des protocoles dont la PCS peut être améliorée;

2. Il doit être possible de mesurer la PCS.

Ces deux idées sont le point de départ de nos premières contributions.

Nous commençons par une description du protocole Signal afin de montrer quelles modifications nous pouvons lui apporter dans le but d'améliorer sa PCS. Cette analyse nous amène à la construction de deux protocoles, MARSHAL et SAMURAI, qui améliorent la PCS de Signal. Nous choisissons d'être le plus proche de Signal afin de permettre une comparaison plus simple de ces protocoles par rapport à Signal. Nous renforçons cette contribution théorique par une implémentation de MARSHAL et SAMURAI permettant d'évaluer leur capacité à être déployés dans un contexte pratique.

L'étape suivante est de formaliser les améliorations que nous avons établies et de vérifier que nos solutions sont effectivement meilleures, en terme de PCS, que le protocole Signal. Notre but n'est pas seulement de considérer ces protocoles mais d'étendre l'étude à d'autres cas, comme SAID (une autre variante de Signal basée sur l'identité) ainsi que sur une suite de procédures de la 5G nommée handover (*i.e.*, procédure relais). Notre approche générique permet de comparer des protocoles qui, a priori, sont incomparables.

Dans la dernière partie de ce manuscrit, nous appliquons les méthodes de la sécurité prouvables dans un autre contexte que précédemment. Le but est de proposer un schéma sûr dans le contexte de l'attestation. Employer des schémas dont la construction est prouvée sûre est une première étape, il faut ensuite garantir qu'une fois déployés, ils garantissent encore une sécurité. L'attestation permet d'avoir une autre garantie, celle d'assurer qu'un composant, qui pourrait être utilisé par un protocole, n'a pas été altéré. Nous proposons donc de formaliser et de construire un schéma sûr pour de l'attestation en profondeur. Notre

modèle est le premier sur ce sujet, ce qui nous permet de prouver, dans le modèle calculatoire, que notre solution est bien sécurisée.