2019 IEEE European Symposium on Security and Privacy (EuroS&P)

SAID: Reshaping Signal into an Identity-Based Asynchronous Messaging Protocol with Authenticated Ratcheting

Olivier Blazy*, Angèle Bossuat†, Xavier Bultel†, Pierre-Alain Fouque†, Cristina Onete* and Elena Pagnin‡
*University of Limoges, XLIM, CNRS UMR 7252
{olivier.blazy, maria-cristina.onete}@unilim.fr
†Univ Rennes, CNRS, IRISA
{angele.bossuat, xavier.bultel, pierre-alain.fouque}@irisa.fr
‡Aarhus University
elena@cs.au.dk

***Abstract*—As messaging applications are becoming increasingly popular, it is of utmost importance to analyze their security and mitigate existing weaknesses. This paper focuses on one of the most acclaimed messaging applications: Signal.**

Signal is a protocol that provides end-to-end channel security, forward secrecy, and post-compromise security. These features are achieved thanks to a key-ratcheting mechanism that updates the key material at every message. Due to its high security impact, Signal's key-ratcheting has recently been formalized, along with an analysis of its security.

In this paper, we revisit Signal, describing some attacks against the original design and proposing SAID: Signal Authenticated and IDentity-based. As the name indicates, our protocol relies on an identity-based setup, which allows us to dispense with Signal's centralized server. We use the identity-based long-term secrets to obtain persistent and explicit authentication, such that SAID achieves higher security guarantees than Signal.

We prove the security of SAID not only in the *Authenticated Key Exchange* (AKE) model (as done by previous work), but also in the *Authenticated and Confidential Channel Establishment* (ACCE) model, which we adapted and redefined for SAID and asynchronous messaging protocols in general into a model we call identity-based Multistage Asynchronous Messaging (iMAM). We believe our model to be more faithful in particular to the true security of Signal, whose use of the message keys prevents them from achieving the composable guarantee claimed by previous analysis.

I. Introduction

Secure asynchronous messaging protocols aim to enable secure-channel establishment between two peers who may not be simultaneously online. Arguably, the most popular and frequently-used such protocol today is Signal, deployed in environments such as the Signal application, WhatsApp, and the *secret conversation* feature of Facebook Messenger. An attractive feature of this protocol, which is introduced as *privacy that fits in your pocket*, is that is has been cryptographically analyzed, and its properties have been formalized and proved [1]. Thus, Signal was shown to provide end-to-end message encryption, implicit entity authentication, forward secrecy, and a form of post-compromise security [2]. The latter is a rare property in secure channel-establishment, and captures the "healing", in time, of the security of a compromised channel. Signal achieves these properties by means of an ingenious mechanism called *key-ratcheting*.

Signal is a public-key protocol, in which the users' public keys are stored and forwarded by a centralized server; the latter is a means of providing trust without certification. Whenever a registered initiator Alice wants to talk to a registered responder, Bob (who is potentially offline), the server must forward Bob's credentials to Alice. In return, Bob will also rely on the server to forward him Alice's correct information. This approach requires the server to be *always* available. In addition to the authenticated channel required at setup for the transfer of the public-key information, users must also each establish a *unilaterally authenticated communication channel* to the server, for each new conversation.

Once the user receives its partner's key information from the server, it can proceed to calculating an initial master secret, from which the first keys can be derived. Afterwards, the user regularly ratchets newly-established secrets in order to generate fresh message keys. The message keys are then used to encrypt messages, authenticate them along with some additional data, which are sent in plaintext.

As observed by Cohn-Gordon *et al.* [1], Signal's ratcheting is *not strongly authenticated*. The existing authentication is implicit and relies on the input used for the ratcheting. This opens easy ways of hijacking sessions or running Denial-of-Service attacks, making the formalization of the security notions unnecessarily difficult and inelegant. Finally, in order to decrypt delayed messages, the receiver must keep the related key-material even after ratcheting them. This is a caveat of the forward-security properties of the keys in Signal, which only applies to receiving parties.

In this paper, we aim to improve Signal's authentication and post-compromise security. We rely on a different trust assumption, replacing Signal's centralized server with an Identity-Based infrastructure. Our main idea is to ensure that at each ratchet the user authenticates in a strong way. The outcome is SAID (for Signal, Authenticated and IDentity-based), a secure

DOI 10.1109/EuroSP.2019.00030

asynchronous messaging protocol that is comparably efficient to Signal and provides better security.

A. Our contributions

We claim two fundamental contributions: a formalization of a security model for asynchronous messaging that is more realistic than the previous ones, and a proposition for a new, efficient asynchronous messaging protocol we call SAID, which is provably secure in our security model. We detail each of these contributions below.

A new security model. Cohn-Gordon et al. [1] have provided a first quantification of the security that Signal achieves. Their model was tailored to the complex computation of the message keys, but without capturing the message transmission. With this separation, one can prove that the message keys are indistinguishable from random by a Man-in-the-Middle adversary. This is a strong, composable security guarantee: it implies that those keys can be securely used for *any* symmetric-key primitive requiring a key of that size.

In Signal, however, the message keys are used for authenticated encryption *with additional data* (AEAD); the users send plaintext information like ratcheting keys as part of the AD. This allows the adversary to trivially distinguish between real and random message keys. Although the message keys can still be used to construct a secure channel, one cannot generalize this property to other symmetric-key properties, so we lose composability. This is similar to the authenticated and confidential channel establishment (ACCE) property, originally defined in the context of TLS 1.2 [3].

We argue that this is not necessarily bad, since Signal's purpose is to construct a secure channel, and define a new, fine-grained security model which formalizes:

Confidentiality: an ACCE-like property for the security of exchanged messages, including forward secrecy;

Authentication: persistent authentication of the two communication partners, which is explicit at each ratchet;

Healing: a degree of post-compromise security which is stronger than for Signal; indeed, compromising the key-chain at a given moment will affect usually fewer or at worst as many key-stages as for Signal.

Our security model, entitled iMAM (for identity-based multistage asynchronous messaging) is tailored to identity-based (IB) protocols and provides fine-grained (adaptive) corruption capabilities. We provide the adversary with three distinct corruption oracles: one that leaks a user's long-term keys, a second one that reveals stage-specific ephemeral information (including the message key if it has been computed by that user), and a third that enables black-box access to computations with the long-term user keys. Our adversaries are unrestricted in their oracle queries; however, some combinations of queries impact the *freshness* of specific stage keys.

We define security in terms of the security of the channel established in various stages (including post-compromise and forward security), and persistent authentication. All these properties are defined per stage, capturing a single ratchet of the keys. The adversary is a Man-in-the-Middle that can query any sequence of oracles, and a freshness notion indicates which stages the adversary can then attack.

A new trust model. Our security definitions are specifically meant to capture protocols in the IB setting. Consequently, we need not take into account Signal's centralized server, but rather consider a Key-Distribution Center (KDC), which generates key-material for all the users. We note that the KDC only needs to be available when a new user requests credentials, not for each conversation.

Cohn-Gordon et al. [1] prove security only in the presence of a semi-trusted centralized server. A malicious server could set up Man-in-the-Middle attacks in each ongoing conversation without being noticed. In our security model, we do not explicitly consider a malicious KDC; however, we do take that scenario into account by giving the adversary the ability to corrupt long-term keys. Thus, our security statements do cover malicious KDCs.

The SAID protocol. As a second main contribution, we propose a new protocol called SAID (for Signal, Authenticated and IDentity-based). Each user is associated in our setting with a unique identity and a long-term key generated by a Key-Distribution Center. The user's identity acts as a public key, allowing an initiator Alice to contact any responder Bob whose identity she knows.

Our protocol splits the communication between the two peers into doubly-indexed stages, like Signal. Symmetric ratchets are used when the same sender chooses to send a new message, while asymmetric ratchets indicate a change of sender. The message keys in SAID are derived from base keys, as in the case of Signal. However, we also bring several non-trivial modifications to that protocol.

Our most fundamental modification is adding persistent authentication (via the user's long-term key) at each ratchet and key-computation. The initial master secret computed during session setup relies on the responder's long-term keys and on randomness generated by the initiator. To explicitly authenticate the latter, the message containing that randomness is signed by the initiator using an IB signature. The master secret will subsequently enter in the computation of every ratchet and KDF-call, authenticating the parties that communicate. Moreover, we add a pseudorandom value into symmetric ratchets, to further improve our post-compromise security.

Authenticating each ratchet and key-computation significantly improves the security guarantees of our protocol over those of Signal. In the latter, compromising a single base-key would result in compromising the entire chain of stages corresponding to subsequent symmetric ratchets. In our case, one requires leakage of both ephemeral *and* long-term secrets to achieve the same goals. Knowledge of *only ephemeral information* yields compromises only a single stage. Knowing *only the long-term key* will compromise the stages until the initiator's first honest ratchet. Even compromising both long-term and ephemeral secrets will only yield information until the first asymmetric ratchet of the user who was sending at

the time of the compromise.

Our protocol requires a single pairing computation for the master secret, a single IB signature, and the two KDFs required by Signal. We formalize and prove the security properties of SAID in the Random Oracle Model (ROM) – as for Signal.

KDC versus Server. An important difference between Signal and SAID lies in the latter's IB setup. Yet, for both protocols, the most dangerous kind of adversary is a corrupted trusted party: for Signal, its centralized server, in our case, the KDC. In that worst case, we lose the same security as Signal in terms of compromised stages; however, for SAID, the KDC learns *user secret keys*, whereas for Signal, the attacker learns nothing but public keys. Consequently for our protocol it is essential that the key used for asynchronous messaging is *not* used in any other application. Moreover, contrary to a corrupted server, a corrupted KDC does not need to be active to corrupt the master secret key, which makes it undetectable from the users' point of view.

Separate leakage queries. The best security for our protocol is obtained when the adversary learns either ephemeral, stage-specific information, or long-term information, but not both. However, in order for that security to be achieved in practice, long-term and ephemeral information must be separated in terms of storage and handling. This could be done by using a trusted execution environment, a secure module, or other such mechanisms. In other words, the security of our protocol – which is much stronger than that of Signal – *can* be achieved, as long as the implementation does not easily allow the adversary access to both ephemeral and long-term data, even if the adversary has access to black-boxes that simulates the functions implemented in the trusted environment.

We stress that our two contributions (the identity-based master-secret generation and its use in the KDFs) are independent. Thus, SAID could use a master-secret generated using public keys as in Signal, resulting into a protocol that achieves the same security.

B. Related Work

Cohn-Gordon *et al.* were the first to formalize the security properties of Signal in 2017 [1]. They proved the properties attained by the underlying key-establishment and ratcheting designs. This analysis – though innovative – does not faithfully capture the way the key-establishment protocol is used in Signal, as we explained in more detail in Section I-A. In this paper, we show that the most that can be proved for those keys is a weaker, non-composable ACCE-like guarantee [3].

Jaeger and Stepanovs [4] very recently studied the security of bidirectional channels against state compromise, introducing both a new security notion and a new construction, which they show is stronger than Signal in their model. In their construction, they define and use key-updatable schemes for encryption and signature. One major difference between Signal and their construction is that their key-updates take the transcript as input, when Signal's ratchets only take the previous key. In particular, this means that they consider the reordering (or dropping) of messages to be an attack, as a message cannot be decrypted unless the previous one was received, whereas Signal views their handling of so-called out-of-order messages as a feature. Even though both sides have valid arguments, we choose to stay close to Signal and allow messages of the same "batch" to arrive in any order.

Authenticated Key-Exchange and Asynchronous Messaging. Cohn-Gordon *et al.*'s [1] groundbreaking paper was followed by an extension of secure asynchronous messaging in group chats [5] and more generic treatments of ratcheted encryption and key-exchange [6], [7]. The focus of the latter works is much larger than that of the original analysis of [1]: instead of simply analyzing a real-world protocol, [6] and [7] formalize (stronger) security requirements, which they argue should be achieved by any ratcheted key-exchange scheme. Subsequently, they instantiate their primitive and prove the security of their constructions. However, both these approaches once more focus on the key-establishment protocol, rather than considering its encapsulation into the messaging mechanism. By contrast, in this paper, we stay close to the full details of the protocols we present. Although subject to different limitations, asynchronous-messaging protocols share many elements of construction and modelling with authenticated key-exchange (AKE) protocols [8]. In this paper, we will combine elements specific to the ACCE security introduced in the context of TLS by Jager *et al.* [3] with aspects of multi-stage AKE security [9]. However, both notions we propose in this paper are adapted from the AKE setting to that of asynchronous messaging, specifically capturing ratcheting mechanisms, out-of-order messaging, and the way the established keys are used.

Identity-based cryptography. This concept was introduced by Adi Shamir [10] with the goal of alleviating the excessive reliance on a public key infrastructure. Assuming the existence of an authority, users are now determined by an identity that is short and easy to remember, *e.g.*, an email address or a phone number. The first (publicly available) instantiations of identity-based encryption [11] were followed by works focusing on identity-based signatures [12] and authenticated key exchange protocols in the AKE model [13]. In this paper, an identity-based signature scheme supersedes Signal's key storage server, bringing the additional advantage of public verification with respect to a known identity (in lieu of a given verification key).

II. Preliminaries

Notations. Identities, *e.g.*, A, P, are binary strings of arbitrary length. The security parameter of cryptographic schemes is denoted by 1^λ. Empty strings of opportune length are denoted with the symbol ε.

Identity-based Signatures. An Identity-Based Signature (IBS) [12] scheme is made up of four possibly randomized algorithms $\mathtt{IBS} = (\mathtt{IBS.Setup}, \mathtt{IBS.Extr}, \mathtt{IBS.Sign}, \mathtt{IBS.Vrfy})$with the following properties:

$\texttt{IBS.Setup}(1^\lambda)$ outputs a master-secret-key $\texttt{IBS}.\mathsf{msk}$ a master publi -key $\texttt{IBS}.\mathsf{mpk}$, and some public parameters $\texttt{IBS}.\mathsf{param}$that are implicit input to all of the following algorithms.

$\texttt{IBS.Extr}(\texttt{IBS}.\mathsf{msk}, I)$: on input the public parameters, the master secret key and an identity $I \in \{0,1\}^*$, the key-extraction algorithm outputs a private signing key $\texttt{IBS}.\mathsf{sk}_I$.

$\texttt{IBS.Sign}(\texttt{IBS}.\mathsf{mpk}, \texttt{IBS}.\mathsf{sk}_I, M)$: on input the public parameters, a user's signing key and a message, the sign algorithm returns a signature sgn.

$\texttt{IBS.Vrfy}(\texttt{IBS}.\mathsf{mpk}, I, M, \mathsf{sgn})$: on input the public parameters, an identity, a message and a signature, the verification algorithm returns 1 (to accept) or 0 (to reject) user P's signature.

In this work, we consider the notion of existential unforgeability against chosen message attacks (EUF-CMA). We consider an adversary $\mathscr{A}$ that receives a master-public-key from some challenger, and that has access to an extraction oracle that simulates the extraction algorithm for chosen identities, and a signature oracle that simulates the signature algorithms for chosen identities and messages. An IBS is said to be EUF-CMA-secure if for any polynomial time $\mathscr{A}$, the probability that $\mathscr{A}$ outputs a fresh signature for an identity that was never queried to the extraction oracle is negligible.

We define the experiment of existential unforgeability against chosen message attacks in Experiment 1, and define the advantage of a probabilistic polynomial time algorithm $\mathscr{A}$ as:

$$\mathsf{Adv}^{\mathsf{EUFCMA}}_{\mathscr{A},P}(1^\lambda) = \mathbb{P}\left[1 \leftarrow \mathbf{Exp}^{\mathsf{EUFCMA}}_{\mathscr{A},P}(1^\lambda)\right].$$

An IBS scheme is said to be EUFCMA-secure if the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{EUFCMA}}_{P}(1^\lambda) = \max_{\mathscr{A}}\left\{\mathsf{Adv}^{\mathsf{EUFCMA}}_{\mathscr{A},P}(1^\lambda)\right\}.$$

Experiment 1 : $\mathbf{Exp}^{\mathsf{EUFCMA}}_{\mathscr{A},P}(1^\lambda)$

1: $\mathsf{s} = (\texttt{IBS}.\mathsf{msk}, \texttt{IBS}.\mathsf{mpk}, \texttt{IBS}.\mathsf{param}) \leftarrow \texttt{IBS.Setup}(1^\lambda)$
2: $\mathcal{O} = \{\texttt{IBS.Extr}(\texttt{IBS}.\mathsf{msk},\cdot), \texttt{IBS.Sign}(\texttt{IBS}.\mathsf{sk}, \texttt{IBS}.\mathsf{mpk}, \cdot)\}$
3: $(\mathsf{sgn}, I, M) \leftarrow \mathscr{A}^{\mathcal{O}}(\mathsf{s})$
4: **if** (I,M) was not sent to the oracle $\texttt{Sign}$ and I was not sent to the oracle $\texttt{Extr}$ and $\texttt{IBS.Vrfy}(\texttt{IBS}.\mathsf{param}, I, M, \mathsf{sgn})$=1 **then**
5: **return** 1 // the adversary wins
6: **else**
7: **return** 0 // the adversary loses
8: **end if**

Authenticated Encryption with Associated Data. An Authenticated-Encryption scheme with Associated Data (AEAD) [14] is made up of three algorithms $\texttt{AEAD} = (\texttt{AEAD.Gen}, \texttt{AEAD.Enc}, \texttt{AEAD.Dec})$ with the following properties:

$\texttt{AEAD.Gen}(1^\lambda)$ outputs the sets of keys $\mathsf{KeySet} \subseteq \{0,1\}^k$, nonces $\mathsf{NonceSet} = \{0,1\}^n$, messages $\mathsf{MsgSet} \subseteq \{0,1\}^*$, and associated data (header) $\mathsf{HeadSet} \subseteq \{0,1\}^*$; where the last two sets have a linear-time membership test.

$\texttt{AEAD.Enc}(K, N, M, \mathrm{AD})$ the encryption algorithm is deterministic and takes as input a key $K \in \mathsf{KeySet}$, a nonce $N \in \mathsf{NonceSet}$, a message $M \in \mathsf{MsgSet}$ and associated data $\mathsf{AD} \in \mathsf{HeadSet}$. It returns the ciphertext $\mathsf{ctx} \in \{0,1\}^*$. For brevity, we often represent this algorithm as $\texttt{AEAD}_K[M, \mathrm{AD}]$.

$\texttt{AEAD.Dec}(K, N, \mathsf{ctx}, \mathrm{AD})$ is a deterministic algorithm that given a key $K \in \mathsf{KeySet}$, a nonce $N \in \mathsf{NonceSet}$, a ciphertext $\mathsf{ctx} \in \{0,1\}^*$, and associated data AD returns either a string in MsgSet or a distinguished symbol $\perp$ (invalid).

In this work, we consider the notion of length-hiding security LH-AEAD introduced by Paterson *et al.* [15]. We consider an adversary $\mathscr{A}$ that interacts with a challenger that picks a bit b at random, and generates a secret key K. The adversary has access to an encryption oracle that takes as input a couple of messages (M_0, M_1) and an additional data AD, and that returns $\texttt{AEAD.Enc}(K, N, M_b, \mathrm{AD})$, and has access to a decryption oracle that takes as input a ciphertext C and an additional data AD, and that returns $\texttt{AEAD.Dec}(K, N, C, \mathrm{AD})$ if $b = 1$ and C was not generated by the encryption oracle, $\perp$ otherwise. An AEAD is said to be LH-AEAD-secure if for any polynomial time $\mathscr{A}$, the probability that $\mathscr{A}$ outputs b is negligently close to $1/2$.

We define the LH-AEAD experiment in Experiment 2. We define the advantage of a probabilistic polynomial time algorithm $\mathscr{A}$ as:

$$\mathsf{Adv}^{\mathsf{LH-AEAD}}_{\mathscr{A},P}(1^\lambda) = \left|\mathbb{P}\left[0 \leftarrow \mathbf{Exp}^{\mathsf{LH\text{-}AEAD}}_{\mathscr{A},P,0}(1^\lambda)\right] - \mathbb{P}\left[1 \leftarrow \mathbf{Exp}^{\mathsf{LH\text{-}AEAD}}_{\mathscr{A},P,1}(1^\lambda)\right]\right|$$

An AEAD is said to be LH-AEAD-secure if the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{LH\text{-}AEAD}}_{P}(1^\lambda) = \max_{\mathscr{A}}\left\{\mathsf{Adv}^{\mathsf{LH\text{-}AEAD}}_{\mathscr{A},P}(1^\lambda)\right\}$$

Experiment 2 : $\mathbf{Exp}^{\mathsf{LH\text{-}AEAD}}_{\mathscr{A},P,b}(1^\lambda)$

1: $\mathsf{set} = (\mathsf{KeySet}, \mathsf{NonceSet}, \mathsf{MsgSet}, \mathsf{HeadSet}) \leftarrow \texttt{AEAD.Gen}(1^\lambda)$
2: $K \xleftarrow{\$} \mathsf{KeySet}$
3: $\mathcal{O} = \{\mathsf{oLoR}.\texttt{AEAD.Enc}_b(K,N,\cdot,\cdot), \mathsf{oLoR}.\texttt{AEAD.Dec}_b(K,N,\cdot,\cdot)\}$
4: $b^* \leftarrow \mathscr{A}^{\mathcal{O}}(\mathsf{set})$
5: **if** $b = b^*$ **then**
6: **return** 1 // the adversary wins
7: **else**
8: **return** 0 // the adversary loses
9: **end if**

Problems and Hardness Assumptions. Our proofs of security rely on standard cryptographic hardness assumptions related to the DH key exchange and bilinear pairings. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p generated by g. Let $G = (\mathbb{G}, p, g)$, we define the following function: $\varepsilon_{\mathsf{CDH}}(1^\lambda) = \max_{\mathscr{D}}\left\{\mathbb{P}[\mathscr{D}(G, g^a, g^b) = g^{ab} : a, b \xleftarrow{\$} \mathbb{Z}_p]\right\}$.

The $\mathsf{oLoR}.\mathtt{AEAD.Enc}_b(K, N, (M_0, M_1), \mathrm{AD})$ oracle

At the first call, set $\mathscr{C} := \emptyset$
$C_0 \leftarrow \mathtt{AEAD.Enc}(K, N, M_0, \mathrm{AD})$
$C_1 \leftarrow \mathtt{AEAD.Enc}(K, N, M_1, \mathrm{AD})$
if $C_0 = \perp \vee C_1 = \perp$ **then**
  return $\perp$
end if
$\mathscr{C} := \mathscr{C} \cup \{C_b\}$
return C_b

The $\mathsf{oRoR}.\mathtt{AEAD.Dec}_b(K, N, C, \mathrm{AD})$ oracle

$M \leftarrow \mathtt{AEAD.Dec}_b(K, N, C, \mathrm{AD})$
if $b = 1 \wedge C \notin \mathscr{C}$ **then**
  return M
end if
return $\perp$

Let $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T = \langle g_T \rangle$ be 3 cyclic groups of the same prime order p. Let e be a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Let $G_B = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g_1, g_2)$, we define: $\varepsilon_{\mathsf{BCDH}}(1^\lambda) = \max_{\mathscr{D}} \left\{ \mathbb{P}[\mathscr{D}(G_B, g_1^a, g_2^b, g_2^c) = e(g_1, g_2)^{abc} : a, b, c \xleftarrow{\$} \mathbb{Z}_p] \right\}$. We use the following cryptographic hardness assumptions. The *Computational Diffie-Hellman* [16] (CDH) states that $\varepsilon_{\mathsf{CDH}}(1^\lambda)$ is negligible. The *Bilinear Computational Diffie-Hellman* [11] (BCDH) states that $\varepsilon_{\mathsf{BCDH}}(1^\lambda)$ is negligible.

III. Overview of the Signal Protocol

In this section, we present a high-level description of the Signal protocol using a somewhat simpler notation than [1]. A summary of the protocol flow is provided in Figure 1.

Signal is a communication protocol for end-to-end encryption in asynchronous communications. It begins with a user, *e.g.*, Bob, who *registers* to a server by providing a unique identifier B (*e.g.*, a phone number) together with: a user public key idpk_B (for which only Bob knows the corresponding secret key), a mid-term pre-key prepk_B, a signature on prepk_B (that can be verified using idpk_B), and a series of ephemeral public keys ephpk_B^i for $i = 1, 2, \ldots, n$.

At any point in time, another registered user, Alice (with identifier A), can *setup a session* with Bob as follows. Alice queries the server for Bob's public key idpk_B, the current[1] mid-term pre-key prepk_B, and one ephemeral key ephpk_B. Alice generates a *master secret* ms_{AB} using her secret key idsk_A, together with idpk_B, prepk_B, ephpk_B, and some randomness. The ms_{AB} is then fed to a Key Derivation Function (KDF), to obtain what is called a *root key* $\mathsf{rk}^{(0)}$ and a *base key*[2] $\mathsf{bk}^{(0,0)}$. The base key is fed to another KDF to produce the next base key $\mathsf{bk}^{(0,1)}$ and the *message key* $\mathsf{k}^{(0,0)}$, which Alice will use to encrypt (and authenticate) her first message. The KDFs are detailed in the next section. The relation between the keys is reported in Formula (1):

$$\mathsf{ms}_{AB} \xrightarrow{\mathsf{KDF}_1} \begin{cases} \mathsf{rk}^{(0)} \\ \mathsf{bk}^{(0,0)} \xrightarrow{\mathsf{KDF}_2} \begin{cases} \mathsf{bk}^{(0,1)} \\ \mathsf{k}^{(0,0)} \end{cases} \end{cases} \tag{1}$$

In addition, Alice generates a random ratcheting secret key $\mathsf{rchsk}_A^{(0)}$ and the corresponding public key $\mathsf{rchpk}_A^{(0)}$. Alice's message to Bob then includes: the (encrypted) first message of the application layer and, in the associated data (AD), the ratcheting public key $\mathsf{rchpk}_A^{(0)}$ together with the information needed for Bob to reconstruct the shared keys. Using the AD, Bob will retrieve Alice's public information from the server, and make sure they match what she sent.

In Signal, each pair of users may only have at most one session together in their entire lifetimes. Therefore, after a session has been initialized, the parties take turns in refreshing the shared secrets by sending new *ratchet keys* which are used to update the root keys (and the base keys of index $(\cdot, 0)$), via the *ratcheting key exchange* mechanism explained below.

A. Key-Ratcheting in the Signal Protocol

Ratcheting is a procedure that securely updates a shared secret in a unidirectional fashion, *i.e.*, given a key, it is possible to derive the *next* key, but not the *previous* one. The key-schedule of Signal requires that each message is encrypted with a different key. Moreover, Alice's encryption keys should be different from and unlinkable to the keys used by Bob to encrypt to Alice. The evolution of the keys is triggered by two factors: changing the message sender (asymmetric ratchet); or the same user sending or receiving a new message (symmetric ratchet).

To keep track of the current *stage* of the key material, we use the two *superscript* indexes (x, y) where: x denotes the number of asymmetric ratchets that have happened before the key was generated[3], and y denotes the current number of symmetric ratchets (for level x).

Symmetric Ratchet. This technique is performed by a user alone, say Alice, and consists in applying a KDF to obtain one new key $\mathsf{k}^{(x,y)}$ per message to be (authenticated) encrypted if Alice is sending messages and Bob is silent, or decrypted, if she is receiving more messages.

The starting point of a symmetric ratcheting is a base key $\mathsf{bk}^{(x,y)}$ obtained via asymmetric ratcheting if $y = 0$, or symmetric ratcheting, if $y > 0$. A one-step symmetric ratcheting, has the following flow, *i.e.*, this is KDF_2:

$$\begin{cases} \left(\mathsf{bk}^{(x,y)}, 0\right) \xrightarrow{\mathtt{HMAC}} \mathsf{bk}^{(x,y+1)} \\ \left(\mathsf{bk}^{(x,y)}, 1\right) \xrightarrow{\mathtt{HMAC}} t \xrightarrow{\mathtt{HKDF}} \mathsf{k}^{(x,y)} \end{cases} \tag{2}$$

Symmetric key ratcheting can be composed sequentially to derive a chain of keys $\mathsf{k}^{(x,0)}, \mathsf{k}^{(x,1)}, \mathsf{k}^{(x,2)}, \ldots$, that authenticate-encrypt or -decrypt an uninterrupted stream of messages by the same user.

[1] Bob will update this key semi-regularly, hence its appellation "mid-term".
[2] This key is called sending/receiving chain key in [1].
[3] Note that all even values of x mean that Alice is the sender, while odd values means she is the $\mathsf{receiver}$.

Asymmetric Ratchet. This technique is performed by a user, say Bob, when changing his role from receiver to sender and vice versa. This mechanism updates the shared secrets common to both Alice and Bob using fresh randomness input by both parties. Cohn-Gordon *et al.* [1] split asymmetric ratchets in two phases, the first from Bob's point of view and the second from Alice's.

Phase I. To perform the x-th asymmetric ratchet, a user, say Bob, generates a random ratcheting secret key $\mathsf{rchsk}_B^{(x)}$ and computes a *new* shared ratchet key $\Delta^{(x)}$ that combines $\mathsf{rchsk}_B^{(x)}$ and the previously received $\mathsf{rchpk}_A^{(x-1)}$, sent by Alice (*e.g.*, in a DH-like fashion). The new root key $\mathsf{rk}^{(x)}$ and base key $\mathsf{bk}^{(x,0)}$ are derived as follows, *i.e.*, this is KDF_1:

$$\left(\mathsf{rk}^{(x-1)}, \Delta^{(x)}\right) \xrightarrow{\text{HKDF}} \begin{cases} \mathsf{rk}^{(x)} \\ \mathsf{bk}^{(x,0)} \end{cases} \tag{3}$$

With $\mathsf{bk}^{(x,0)}$, Bob can perform a symmetric key ratcheting to generate $\mathsf{k}^{(x,0)}$ and authenticate-encrypt his messages to Alice. Bob shall also include $\mathsf{rchpk}_B^{(x)}$ in the AD of every level x message to Alice. Several symmetric ratchets may happen, until Alice comes online and wishes to reply, for which she needs to contribute with her own randomness and start the next asymmetric ratchet.

Phase II. Upon receiving $\mathsf{rchpk}_B^{(x)}$, Alice can compute the shared ratchet key which she will use, along with the root key $\mathsf{rk}^{(x-1)}$, to derive $\mathsf{rk}^{(x)}$ and $\mathsf{bk}^{(x,0)}$ via (3). Then, after decrypting the messages, in order to reply to Bob, Alice generates a new random ratcheting secret key $\mathsf{rchsk}_A^{(x+1)}$, thus performing the $x+1$-th asymmetric ratchet, and computes a *new* shared ratchet key $\Delta^{(x+1)}$ that combines $\mathsf{rchsk}_A^{(x+1)}$ and $\mathsf{rchpk}_B^{(x)}$. She then derives the new root key $\mathsf{rk}^{(x+1)}$ and her base key $\mathsf{bk}^{(x+1,0)}$ as:

$$\left(\mathsf{rk}^{(x)}, \Delta^{(x+1)}\right) \xrightarrow{\text{HKDF}} \begin{cases} \mathsf{rk}^{(x+1)} \\ \mathsf{bk}^{(x+1,0)} \end{cases} \tag{4}$$

She can then perform symmetric ratchets. Alice will include $\mathsf{rchpk}_A^{(x+1)}$ in the AD of every level $x+1$ message to Bob.

Remark. In [1], stages are counted differently: level x starts when Bob sends his new ratchet key, and ends once he sends the next; with Alice sending her randomness in between. This means that there can be two separate stages of index (x, y): one where Bob sends his y-th message for level x, and one where Alice does. Note that in our model, $\mathsf{rchpk}_B^{(x)}$ exists only for odd values of x, while $\mathsf{rchpk}_A^{(x)}$ exists only for even x. The two models are equivalent, and our choice is motivated by the need to lighten the sub- and superscripts, to simplify the notations.

B. On the Security of the Signal Protocol

Cohn-Gordon *et al.* [1] performed the first formal analysis of Signal as an authenticated key-exchange protocol. They showed that under standard cryptographic assumptions, Signal provides implicit AEAD key-authentication, forward secrecy and, if used correctly, a form of post-compromise security. The authentication of the keys is implicit, since it is derived from the fact that only the intended party could compute the key; however, Signal gives no explicit guarantee that the intended party actually did compute the key. Forward secrecy assures that, if at a certain point in time t^* an adversary corrupts a party, it is still impossible for the adversary to decrypt a message sent at any time $t < t^*$. In particular, all ratchet keys used before the moment of corruption remain secure. Post-compromise security is a healing property: after a party has been corrupted at time t^* the key material at time t^* is no longer secure, but if the party performs an honest asymmetric ratcheting, all the subsequent keys (produced at any time $t > t^* + \delta$) are again secure.

IV. Some problems in Signal

Despite its innovative features and good security guarantees [1], Signal *does* have some weaknesses. In this section, we present some problems in the design of the Signal protocol; we defer our mitigations to Section V. We remark that the threats described below fall outside the security model considered in [1] and come from looking at Signal from a new perspective: Cohn-Gordon *et al.* aimed to show which security properties Signal *does* guarantee, we focus on possible flaws and new, stronger proposals.

Symmetric Ratcheting. In Signal, as long as Alice continues to send messages to Bob without receiving a reply, the sending-key chain grows via symmetric ratcheting. In this setting, an attacker that manages to expose one base key will be able to learn any future base- and message-keys in that chain. In particular, all future messages sent from the moment of the exposure, and until the next asymmetric ratcheting, can be decrypted by the attacker.

Session Hijacking. When performing asymmetric ratcheting, the two parties contribute with fresh Diffie-Hellman elements that are authenticated only by the current message keys. Thus, an attacker that can expose Alice's current ephemeral state can hijack her session from the *next* asymmetric ratchet. Concretely, the attacker chooses what fresh ratcheting information to send to Bob, and will be able to derail Bob on a new track of key-chains that diverges from the honest one, and impersonate Alice in the long run, even without knowing her secret key.

An Online Credential Server. Signal's session initialization heavily relies on a server that stores and forwards users' public data. Consequently, this server must be online at all times and users must trust the long-term credentials of their partners provided by the server. While users' credentials are not confidential information, the fact that it is sent without authentication from the server's side gives room for Man-in-the-Middle attacks that are undetectable by the users.

Out-of-order Messages. In asynchronous messaging, the order in which Alice sends messages might be different from the order in which Bob receives them. When this happens, Bob advances his chain of keys to keep up with the received messages' states and stores the keys related to the pending (not yet delivered) messages. In such a scenario, an attacker

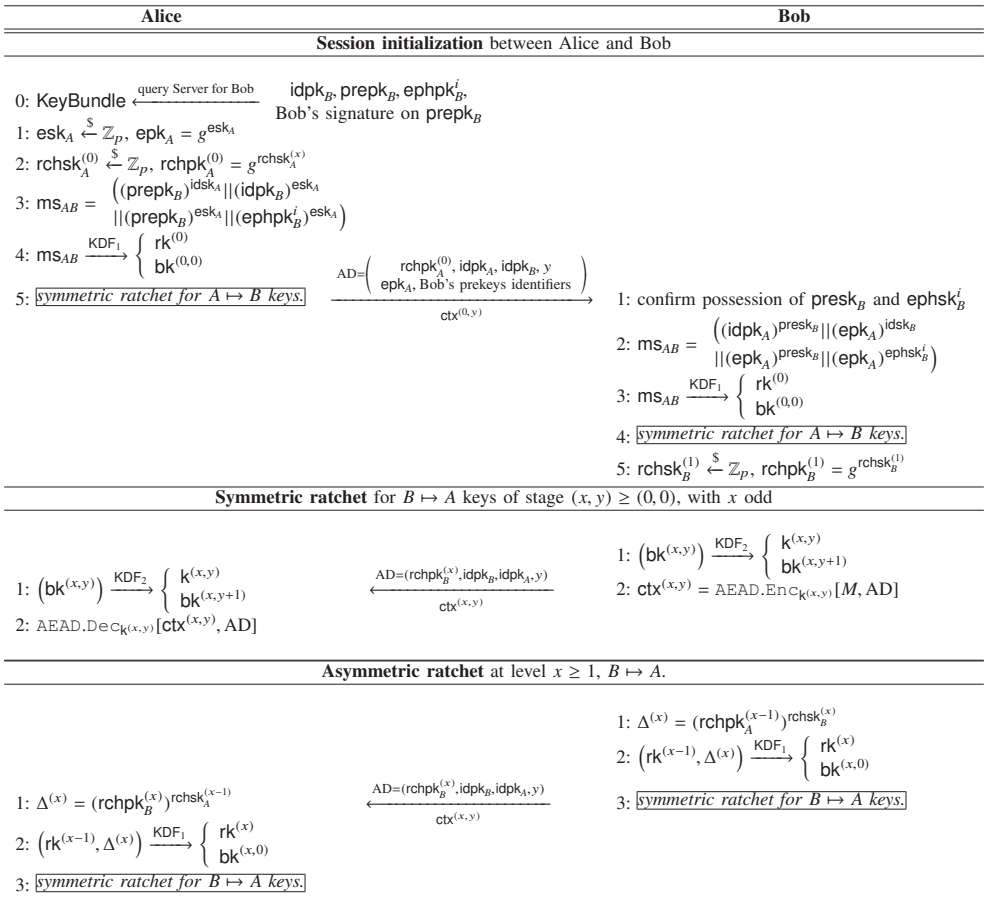

Fig. 1: Protocol flow of the core parts of Signal [1] using the notation adopted in this paper, where g is the generator of a group of prime order p.

that retains all of Alice's messages apart from the *last* one, and then exposes Bob's current state, is able to decrypt all the withheld ciphertexts using Bob's stored key-chain.

V. SAID

We present now the first of our main contributions, the SAID protocol. Our aim is to stay as close as possible to the original Signal protocol while mitigating some of the threats discussed in Section IV. Concretely, we make three major changes.

1) We replace the semi-trusted credential server with a Key-Distribution Center (KDC) that provides identity-based secret keys for all the users. In this way, the KDC has to be online at each user-registration and no longer at any session-setup. In addition, the identity-based infrastructure rules out the chance of Man-in-the-Middle attacks.
2) We upgrade the key ratcheting mechanism to include long term identity-based secrets that guarantee stronger and explicit partner authentication.
3) Finally, we introduce the use of a trusted execution environment to securely implement the execution of functions on sensitive data.

The SAID protocol has four main phases (more details in the following sections):

Parameter Generation run once, by a trusted party, to set up the public parameters of the protocol.

User Registration performed by users at installation time (and subsequently periodically) to create their identity-based cryptographic data in the KDC.

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 that will be shared with B only (see top of Figure 2 for details).

Messaging takes place when two users communicate in a session. This phase is characterized by sequences of symmetric and asymmetric ratchets (see bottom of Figure 2 for details).

A. Parameter Generation

The KDC sets up its master secret keys and the public parameters used by SAID as follows.

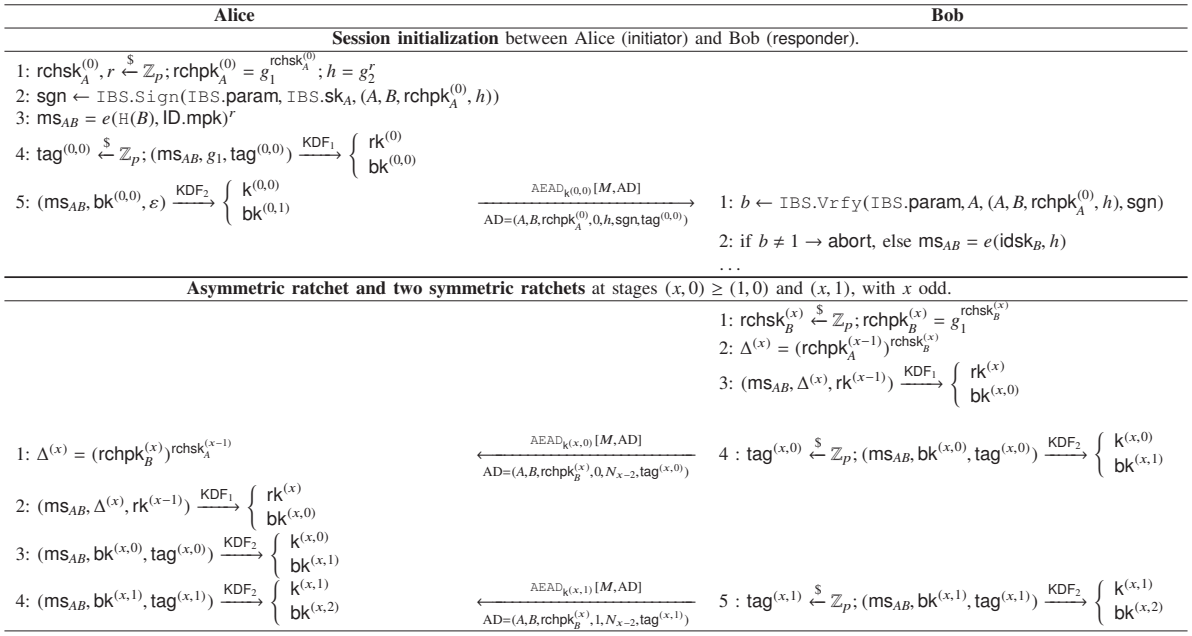

Fig. 2: Session initialization (above) and ratchets (below) in SAID, where N_{x-2} denotes the number of messages that the sender sent at level $x-2$, and M denotes the plaintext of the current stage.

- $\mathtt{AEAD.param} = (\mathsf{KeySet}, \mathsf{NonceSet}, \mathsf{MsgSet}, \mathsf{HeadSet})$, obtained from $\mathtt{AEAD.Gen}(1^\lambda)$,
- $(\mathtt{IBS.param}, \mathtt{IBS.mpk}, \mathtt{IBS.msk}) \leftarrow \mathtt{IBS.Setup}(1^\lambda)$,
- a description of a DH bilinear mapping $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$,
- a random secret master key for handling identities, $\mathsf{ID.msk} \xleftarrow{\$} \mathbb{Z}_p$, and a corresponding master public key $\mathsf{ID.mpk} = g_2^{\mathsf{ID.msk}} \in \mathbb{G}_2$,
- a description of a hash function $\mathtt{H} : \{0,1\}^* \rightarrow \mathbb{G}_1^*$.
- a description of two Key Derivation Functions KDF_1 and KDF_2. The first KDF is used to generate the next root- and base-key $\mathsf{KDF}_1 : \mathbb{G}_T \times \mathbb{G}_1 \times \{0,1\}^{\mathsf{size(bk)}} \rightarrow \{0,1\}^{\mathsf{size(rk)}} \times \{0,1\}^{\mathsf{size(bk)}}$ and is defined as in Signal. The other one is used to generate the next key for AEAD and base-key: Let $\mathtt{HMAC} : \{0,1\}^* \rightarrow \{0,1\}^{\mathsf{size(bk)}}$ and $\mathtt{HKDF} : \{0,1\}^{\mathsf{size(bk)}} \rightarrow \{0,1\}^{\mathsf{size(k)}}$ be two hash functions, $\mathsf{KDF}_2 : \mathbb{G}_T \times \{0,1\}^{\mathsf{size(bk)}} \times \{0,1\}^{\mathsf{size}(p)} \rightarrow \{0,1\}^{\mathsf{size(k)}} \times \{0,1\}^{\mathsf{size(bk)}}$ is defined as follows: $\mathsf{KDF}_2(x, y, z) \rightarrow (\mathtt{HKDF}(\mathtt{HMAC}(x||y||z)), \mathtt{HMAC}(x||y))$.

SAID public parameters set pparam includes all the public parameters $\mathtt{AEAD.param}$, $\mathtt{IBS.param}$, $\mathtt{IBS.mpk}$, $\mathcal{G}$, $\mathsf{ID.mpk}$, and all the descriptions of H, KDF_1, and KDF_2. The master secrets, kept by the KDC only, are $\mathtt{IBS.msk}$ and $\mathsf{ID.msk}$.

B. 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 $\mathtt{IBS.sk}_A \leftarrow \mathtt{IBS.Extr}(\mathtt{IBS.param}, \mathtt{IBS.msk}, A)$ and her secret identification key $\mathsf{idsk}_A \in \mathbb{G}_1$ generated as[4] $\mathsf{idsk}_A = \mathtt{H}(A)^{\mathsf{ID.msk}}$. 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'.

C. Session Initialization

In SAID any registered user A can initiate a session with another registered user B (without the KDC being online), following the procedure depicted on the top of Figure 2.

In detail, A chooses a random ratchet secret key, $\mathsf{rchsk}_A^{(0)}$, and computes its corresponding DH public key $\mathsf{rchpk}_A^{(0)}$. As it is 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 r and computes $h = g_2^r$. At this point A signs $\mathsf{sgn} \leftarrow \mathtt{IBS.Sign}(\mathtt{IBS.mpk}, \mathtt{IBS.sk}_A, d = (A, B, \mathsf{rchpk}_A^{(0)}, h))$ where the identities are included in the signed message to avoid replay attacks. The values h and sgn will be part of the AD of any level-0 message sent to B. The master secret shared between A and B is $\mathsf{ms}_{AB} = e(\mathtt{H}(B), \mathsf{ID.mpk})^r$. To generate the initial root key $\mathsf{rk}^{(0)}$ and base-key $\mathsf{bk}^{(0,0)}$ the values $(\mathsf{ms}_{AB}, g_1, \varepsilon)$ are input to KDF_1. With $\mathsf{bk}^{(0,0)}$, A can perform the first *symmetric ratchet*, *i.e.*, she generates a random $\mathsf{tag}^{(0,0)}$ and computes $\mathsf{KDF}_2(\mathsf{ms}_{AB}, \mathsf{bk}^{(0,0)}, \mathsf{tag}^{(0,0)})$ to obtain the first AEAD key $\mathsf{k}^{(0,0)}$ together with the next base-key $\mathsf{bk}^{(0,1)}$. Finally, A authenticate-encrypts the message M with $\mathsf{AD} = (A, B, \mathsf{rchpk}_A^{(0)}, 0, h, \mathsf{sgn}, \mathsf{tag}^{(0,0)})$, and sends $\mathtt{AEAD}_{\mathsf{k}^{(0,0)}}[M, \mathsf{AD}]$ to B.

[4]The user's secret identification key is essentially a Boneh-Franklin key for identity-based encryption [11].

For the session initialization to be successful, the responder (paired user B) needs to reply to A's message *consistently*. This happens only if both of the following conditions hold true:

1) $1 = \texttt{IBS.Vrfy}(\texttt{IBS}.\mathsf{param}, A, d, \mathsf{sgn})$, *i.e.*, A's signature verifies for identity A; and
2) $\mathsf{ms}_{AB} = e(\mathsf{idsk}_B, h)$, *i.e.*, if A and B generate the same master secret (and consequently the same encryption/decryption key $\mathsf{k}^{(0,0)}$).

D. 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 base- 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 y counter of the chat state, so if the starting stage is (x, y), after the symmetric ratchet we land at stage $(x, y + 1)$.

The process of a symmetric ratchet is depicted in the lower part of Figure 2, lines 3 and 4 for Alice, or 5 and 7 for Bob. In a nutshell, A inputs the shared master secret, the current base key (of stage (x, y)), and a fresh random tag to KDF_2 and obtains the authentication-encryption key of stage (x, y) and the next base-key for stage $(x, y + 1)$. Note that, as it is in Signal, KDF_2 is split in two parts, as shown in Formula 5: one that generates the next base-key, and one that generates the encryption key – only the latter uses the random tag as input, in order to handle out-of-order messages, *i.e.*, the base-keys could be computed simply from the previous ones, but not the encryption keys.

$$\begin{cases} \left(\mathsf{ms}_{AB}, \mathsf{bk}^{(x,y)}\right) \xrightarrow{\texttt{HMAC}} \mathsf{bk}^{(x,y+1)} \\ \left(\mathsf{ms}_{AB}, \mathsf{bk}^{(x,y)}, \mathsf{tag}^{(x,y)}\right) \xrightarrow{\texttt{HMAC}} t \xrightarrow{\texttt{HKDF}} \mathsf{k}^{(x,y)} \end{cases} \tag{5}$$

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

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 x counter and resets the y counter of the chat state, so if the starting stage is (x, y), after the asymmetric ratchet we land at stage $(x + 1, 0)$.

The process of asymmetric ratcheting is depicted in the lower part of Figure 2, lines 1 and 2 for Alice, or 1 to 3 for Bob. Assume that A has sent the last message (which includes her level $x-1$ ratchet key $\mathsf{rchpk}_A^{(x-1)}$ in AD), then, to send his response, B selects a random ratchet secret key $\mathsf{rchsk}_B^{(x)}$ and computes the DH shared secret $\Delta^{(x)} = (\mathsf{rchpk}_A^{(x-1)})^{\mathsf{rchsk}_B^{(x)}}$. He then inputs the shared master secret, the newly computed DH secret, and the current root key (of level $x - 1$) to KDF_1 and obtains the level x root key together with the new base-key for stage $(x, 0)$. Finally, B performs a symmetric ratchet to generate the authentication-encryption key of stage $(x, 0)$ (and the next base-key for stage $(x, 1)$).

Note that, furthermore, as depicted in Figure 2, the additional data sent along with the message at stage (x, y) contains Alice and Bob's identity, the level x ratchet public key of the current sender, the index counter y, the number N_{x-2} of messages that the sender sent at level $x - 2$ (set to 0 for N_{-1} and N_{-2}), and, finally, the tag $\mathsf{tag}^{(x,y)}$.

E. Long-term secret key in SAID.

Signal requires private keys with different security needs, *i.e.*, ephemeral, mid-term and long-term keys. However, in the AKE security model of [1], authors do not distinguish these different levels of security, in the sense that each key can be corrupted in the same way by the adversary using the so-called "reveal" oracles. In SAID, we assume that for any pair of user A, keys sk_A and ms_{AB} (for each user B that interacts with A) are long-term keys, and we define a security model that provides a fine-grained analysis of the long-term keys compromise, following the definition of Cohn-Gordon in [2]. More precisely, we distinguish three level of compromise:

Total compromise: the adversary learns the key sk_A or ms_{AB}. In this case, SAID has the same security as Signal, because the adversary can run the key derivation functions on her own, as in Signal.

Weak compromise: the attacker does not have knowledge of the key, however she can compute operations that use the long-term keys sk_A and ms_{AB}. More precisely, she has access to a compromise oracle that simulates all routines that require these keys over a given period of time. Intuitively, the oracle can be used for signing a given message using the user's secret key, and for computing key derivation functions on the master secret ms_{AB} and some input values. The routines simulated by the oracle are detailed in Algorithms 3, 4 and 5. In this case, the adversary cannot deduce the keys of the next symmetrical ratchets because she cannot predict what random tags will be chosen the future, so she cannot pre-compute the future message keys using the oracle.

No compromise: the conversation is fully secure, as every message key depends on the long-term secret keys.

In practice, the weak compromise requires some hardware hypothesis, as the long-term keys and the corresponding routines must be implemented in a trusted module called a *Hardware Security Module* (HSM), *e.g.*, a *Trusted Execution Environment* (TEE) [17]. On a smartphone, the sim card can also play the role of the HSM [18]. Another solution is to use a trusted proxy that implements the routines. For instance, the user sends messages using SAID on his smartphone, and he interacts with the proxy to run the sensitive routines. Moreover, the HSM can be simulated by *Trusted Platform Modules* (TPM)[5], which are software modules with equivalent security guarantees.

[5]The International Standard for the TPM is given in https://www.iso.org/standard/66510.html

Algorithm 3 Routine $\mathsf{R}.\mathtt{IBS.Sign}(\mathtt{IBS}.\mathsf{param}, \mathtt{IBS}.\mathsf{sk}_A, \cdot)$:

on input m;
return $\sigma \leftarrow \mathtt{IBS.Sign}(\mathtt{IBS}.\mathsf{param}, \mathtt{IBS}.\mathsf{sk}_A, m)$

We assume that the trusted environment securely stores the master secret keys of every instance of a user. However, SAID can also be securely implemented using an environment without this feature. In this case, the idea is to have r be the hash of the concatenation of the user's secret key and the identity of the user's partner, instead of picking it randomly. More formally, that means replacing the $r \xleftarrow{\$} \mathbb{Z}_p$ instruction by $r \leftarrow \mathtt{H}'(\mathsf{idsk}_A, B)$, where $\mathtt{H}'$ is a hash function in Algorithm 4. Hence, the initiator of a chat can re-compute r on-demand without the need to adaptively store data (ms_{AB}) in the trusted environment. However, the responder must compute the pairing $e(\mathsf{idsk}_B, h)$ in the HSM to retrieve ms_{AB} at each stage, which obviously impacts the protocol's efficiency. An other more efficient solution is to store the encryption of each ms_{AB}, then to send it to the HSM at each query.

F. Performances

In this section, we show that Signal and SAID have equivalent computational cost. First, we remark that SAID has the same complexity as Signal, except at the initialization phase, which is run only once per instance. To compare the initialization cost of Signal and SAID, we give the number of exponentiations, paring computations, and signature computations for both protocols, which are the dominant operations. The sender's initialization algorithm requires 3 exponentiations on a pairing friendly, prime-order group, 1 pairing computation, and 1 identity-based signature generation; and the receiver's initialization algorithm requires 1 exponentiation, 1 pairing computation and 1 identity-based signature verification. By using the certificate based IBS given in [12] instantiated with the Shnorr signature [19], the signature algorithm requires 1 exponentiation, and the verification algorithm requires 4 exponentiations. By instantiating our pairing as in [20], a Tate pairing computation costs approximately 4 times the cost of an exponentiation[6] in $\mathbb{G}_1$. To sum up, initialization cost of SAID is equivalent to 8 exponentiations for the sender, and 9 for the receiver. On the other hand, the Signal initialization algorithm requires 6 exponentiations for the sender, and 5 for the receiver.

VI. Identity-Based Multi-Stage Asynchronous Messaging Protocols

Before defining and modeling Multi-Stage Asynchronous Messaging Protocols in Definition 1, we set the context by formalizing the notions of stages, roles, and party instance.

[6]For instance, for a security of 256 bits, a Tate pairing costs $8726 \cdot \boldsymbol{M}$ and the exponentiation costs $2863 \cdot \boldsymbol{M}$, where $\boldsymbol{M}$ is the multiplication cost in a field and is approximately $13,000$ clock cycles [20].

Algorithm 4 (initiator) Routine $\mathsf{R}.\mathsf{KDF}_*(\mathsf{ms}_{AB}, \cdot)$:

on input (x_1, x_2);
if this is the first call to this routine **then**
  $r \xleftarrow{\$} \mathbb{Z}_p$; $h = g_2^r$; $\mathsf{ms}_{AB} = e(\mathtt{H}(B), \mathsf{ID.mpk})^r$
end if
$y \leftarrow \mathsf{KDF}_*(\mathsf{ms}_{AB}, x_1, x_2)$
return (h, y)

Algorithm 5 (responder) Routine $\mathsf{R}.\mathsf{KDF}_*(\mathsf{ms}_{AB}, \cdot, [h])$:

on input (x_1, x_2);
if this is the first call to this routine **then**
  $\mathsf{ms}_{AB} = e(\mathsf{idsk}_B, h)$
end if
return $y \leftarrow \mathsf{KDF}_*(\mathsf{ms}_{AB}, x_1, x_2)$

A. Groundwork

Parties and roles. We consider a system made of multiple parties. Each party P is associated with a unique party identifier: its identity. By abuse of notation, we use P to denote both the party and its identity. In the spirit of Signal, we simplify the model by allowing each party P to only have one *single* protocol session with each other party throughout their entire lifetimes. We denote the protocol session (or *instance*) between P and Q as π_P^Q, if seen from P's point of view, and π_Q^P, from Q's side. The party that begins the session is called initiator while its partner is called responder.

Stages and Execution Time-Line. In multi-stage protocols, the key material evolves according to the stage the conversation is at. To keep track of this process, stages are defined using an ordered pair of non-negative integers $s := (x, y)$. The first index, x, tells who is currently speaking: even values indicate the initiator of the conversation; odd values indicate the responder. The second index, y, counts how many messages the party that is the current sender has already sent since it started speaking (again).

Each stage has one sender and one receiver, *i.e.*, there is one and only one role per user, and both users cannot have the same role. Stages can evolve in two different ways:

- the current sender sends an additional message, in which case, stage (x, y) turns into stage $(x, y + 1)$;
- there is a switching of sender, and stage (x, y) turns into stage $(x + 1, 0)$.

We denote by $\mathsf{next}(s)$ the *subsequent stages* to a stage $s = (x, y)$, thus $\mathsf{next}(s) = \{(x, y + 1), (x + 1, 0)\}$. For example, for $s = (0, 0)$ we have $\mathsf{next}(0, 0) = \{(0, 1), (1, 0)\}$, and as the initiator keeps sending messages without getting a reply the y index increases: $(0, 1), (0, 2)$, etc. The responder's first reply triggers the transition from stages $(0, \cdot)$ to stage $(1, 0)$. Stages are ordered lexicographically. For practical purposes we assume the protocol implicitly defines a maximal value $y_{\mathsf{max}} > 0$ of messages that a party can send in a row.

B. Syntax

Our definition of Identity-Based Multi-Stage Asynchronous Messaging protocols (iMAM for short) is meant to be general and does not tailor specifically to our SAID proposal.

Let π_P^Q be the (unique) protocol instance between P and Q. For iMAM protocols, we define instances as handles of attributes that can be either *static* (*i.e.*, once set, the value does not change for the whole duration of the session), or *non-static* (*i.e.*, the value is updated during the protocol run, usually at every stage).

- A partner identifier $\pi_P^Q.\mathsf{pid}$, consisting of the identity Q of the intended party partnered in the protocol session. This attribute is *static*.
- A session identifier $\pi_P^Q.\mathsf{sid}$, consisting of the concatenation of the identity of the initiator and the responder, as well as some protocol-specific auxiliary information aux, (P, Q, aux).
- A stage list $\pi_P^Q.\mathsf{stages}$, consisting of a list of "booleans" such that $\pi_P^Q.\mathsf{stages}[s] \in \{1, \perp\}$. We have $\pi_P^Q.\mathsf{stages}[s] = 1$ if and only if a message was sent or received (correctly) at stage s, otherwise $\pi_P^Q.\mathsf{stages}[s]$ doesn't "exist", *i.e.*, is $\perp$. By abuse of notation, we say $s \in \pi_P^Q$ if $\pi_P^Q.\mathsf{stages}[s] = 1$. This attribute is *non-static*.
- A transcript $\pi_P^Q.Tr$, consisting of a list of ordered data $\pi_P^Q.Tr[s] = D$. For clarity, $\pi_P^Q.Tr$ contains all data sent and received by party P during its chat with Q. This attribute is *non-static*.
- A reception indicator list $\pi_P^Q.\mathsf{rec}$, consisting of a list of integers such that $\pi_P^Q.\mathsf{rec}[s] \in [\![0, y_{\mathsf{max}}]\!] \cup \{\perp\}$, if P is the receiver for the stage $s = (x, y)$, then $\pi_P^Q.\mathsf{rec}[s] = y_s$ is such that the message of index (x, y_s) is the first message of level x that P received[7], and if P did not yet receive a message at level x, or is the sender, then $\pi_P^Q.\mathsf{rec}[s] \text{ is } \perp$.
- Several lists of ephemeral elements $\pi_P^Q.\mathsf{var}$, such that $\mathsf{var} \in \mathscr{V}$ where $\mathscr{V}$ is the set of (kinds of) ephemeral elements. For example, in SAID, we define $\mathscr{V} = \{\mathsf{k}, \Delta, \mathsf{bk}, \mathsf{rk}, \mathsf{rchpk}, \mathsf{rchsk}, \mathsf{tag}\}$. Each $\pi_P^Q.\mathsf{var}$ is an ordered list (indexed by stage) of the var elements that are specific to each stage $s \in \pi_P^Q.\mathsf{stages}$[8]. This attribute is *non-static* and can be updated in an append-only fashion. In practice, however, ephemeral elements that are no longer needed are removed from the list. Note that $\pi_P^Q.\mathsf{rchpk}[(x, y)]$ is Q's latest ratchet public key, either sent at level $x - 1$ (if P is the current sender) or level x (if Q is the sender). Similarly, $\pi_P^Q.\mathsf{rchsk}[(x, y)]$ is the latest ratchet private key of P, either created at level $x - 1$ (if P is the current receiver) or level x (if Q is the receiver).

[7]This list exists because we can handle out-of-order messages, and therefore it is likely that $y_s \neq 0$. In particular, the ratchet public key contained in the AD of the message of stage (x, y_s) is the one used for the x-th asymmetric ratchet, independently of the value contained in the additional data of the other level x messages.

[8]For example, in SAID, $\pi_P^Q.\mathsf{rk}[(x, y)]$, *i.e.*, $\mathsf{var} = \mathsf{rk}$, is $\mathsf{rk}^{(x)}$.

Definition 1 (Identity-Based Multi-Stage Asynchronous Messaging Protocols): An *Identity-Based Multi-Stage Asynchronous Messaging Protocol* (iMAM for short) is a tuple of five algorithms iMAM = (`aSetup`, `aUReg`, `aStart`, `aRKGen`, `aSend`, `aReceive`), such that:

- $\mathtt{aSetup}(1^\lambda) \to (\mathsf{msk}, \mathsf{mpk})$: on input a security parameter (in unary) 1^λ, the system setup algorithm outputs a master secret key msk and a master public key[9] mpk.
- $\mathtt{aUReg}(P, \mathsf{msk}) \to \mathsf{ltkeys}_P$: on input a user identity P, and the master secret key msk, the user registration algorithm outputs the user-specific (long-term) secret keys ltkeys_P and a user identifier P.
- $\mathtt{aStart}(\mathsf{ltkeys}_P, \mathtt{role}, Q) \to (\pi_P^Q)$: on input a user P's long term keys ltkeys_P, a $\mathtt{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$ for the party P, and the intended partner's identity $Q \neq P$, the start-a-conversation algorithm initializes and outputs the protocol instance π_P^Q.
- $\mathtt{aRKGen}(1^\lambda) \to (\mathsf{rchsk}, \mathsf{rchpk})$: on input a security parameter (in unary) 1^λ, the ratchet key generation algorithm outputs a public/private pair of ratchet keys (rchsk, rchpk).
- $\mathtt{aSend}(\mathsf{ltkeys}_P, s, \pi_P^Q, \mathsf{M}, [\mathsf{rch}, \mathsf{ms}_{PQ}]) \to (\pi_P^Q, \mathsf{C}, [\mathsf{ms}_{PQ}])$: on input a user's long-term secret keys ltkeys_P, an instance π_P^Q, a stage $s = (x, y)$, a message M, if $s \neq (0, 0)$, a master secret ms_{PQ}, and, if $x = 0$, a public/private ratchet key pair $\mathsf{rch} = (\mathsf{rchsk}, \mathsf{rchpk})$, the send algorithm outputs an updated instance π_P^Q, a new message C (usually a ciphertext with AD), and, if $s = (0, 0)$, a master secret key ms_{PQ}. The behavior of this algorithm highly depends on the input stage s and the party instance π_P^Q.
- $\mathtt{aReceive}(\mathsf{ltkeys}_P, s, \pi_P^Q, \mathsf{C}) \to (\pi_P^Q, \mathsf{M}, [\mathsf{ms}_{PQ}])$: on input a party's secret keys ltkeys_P, an instance π_P^Q, a stage $s = (x, y)$, and a message C (usually a ciphertext), the receive algorithm outputs an updated instance π_P^Q, and a message M (usually the decryption of C).

SAID as iMAM**.** In the case of SAID, the protocol setup `aSetup` outputs the parameters explained in section V-A. In particular, mpk includes the public parameters of an identity-based signature scheme, the description of the prime order groups, a hash function, two key derivation functions, and algorithms of an `AEAD` scheme. User P's long-term secret keys, as returned by algorithm `aUReg`, are one identity secret key idsk_P and one identity signing key $\mathtt{IBS}.\mathsf{sk}_P$. The `aStart` algorithm initializes an empty instance π_P^Q. Note that the party instance π_P^Q is storing all the keys necessary to run the algorithms correctly as explained in Section VI-B. The `aSend` algorithm is defined according to the input stage s: if $s = (0, 0)$ it runs the asymmetric ratchet procedure depicted in Figure 2; if $s = (x, y)$ for some $y > 0$, it runs the symmetric ratchet procedure depicted in Figure 2; if $s = (x, 0)$ for some $x > 0$, it runs the asymmetric ratchet procedure depicted in the Figure

[9]This key is input to all the subsequent algorithms and contains a value $y_{\mathsf{max}} > 0$ that bounds the maximal horizontal growth of stages.

2. In all cases, aSend computes $\mathsf{C} \leftarrow \mathtt{AEAD.Enc}_{\mathsf{k}(s)}[\mathsf{M}, \mathsf{AD}]$ where the additional data contains the value N_{x-2} corresponding to the *number of messages sent by the party last time it was acting as* sender (N_{-2} and N_{-1} are set to 0 by default). The aReceive algorithm is defined exactly as aSend except that instead of encrypting messages, it decrypts ciphertexts, *i.e.*, $\mathsf{M} \leftarrow \mathtt{AEAD.Dec}_{\mathsf{k}(s)}[\mathsf{C}, \mathsf{AD}]$. Finally, the aRKGen algorithm returns a ratchet key pair made up of a random element $\mathsf{rchsk} \xleftarrow{\$} \mathbb{Z}_p$ along with $\mathsf{rchpk} = g_1^{\mathsf{rchsk}} \in \mathbb{G}_1$.

Correctness. In what follows, we define the notion of correctness for iMAM protocols. Loosely speaking, a protocol is correct if, for any pair of users P and Q that run the protocol honestly, any message sent at any stage by P will be correctly decrypted and authenticated by the user Q. Note that our definition considers out-of-order message delivery, *i.e.*, the messages are correctly decrypted even if they are received out-of-order by the user Q.

Definition 2 (correctness): An Identity-Based Multi-Stage Asynchronous Messaging Protocol is *correct* if for any pair of distinct honest users (P, Q), given the party instances π_P^Q and π_Q^P, let $R[s] \in \{P, Q\}$ (resp. $S[s]$) denote the receiver (resp. sender) at a given stage s, for any stage $s = (x, y)$ such that:

- if $x \neq 0$, there exists $s' = (x-1, y') \in \pi_P^Q \cap \pi_Q^P$, *i.e.*, at least one message was sent *and* received at level $x-1$, meaning both parties know the previous ratchet key,
- if $y \neq 0$ then $(x, y-1) \in \pi_{S[s]}^{R[s]}$, *i.e.*, a message was sent at the previous stage,

then, for:

- any message M, and
- any $\mathsf{rch}^{(x)} = (\mathsf{rchsk}_{S[s]}^{(x)}, \mathsf{rchpk}_{S[s]}^{(x)})$ generated by $\mathtt{aRKGen}(1^\lambda)$, and
- any 3-uple $(\pi_{S[s]}^{R[s]}, \mathsf{C}, [\mathsf{ms}_{PQ}])$ generated by the algorithm $\mathtt{aSend}(\mathsf{ltkeys}_{S[s]}, s, \pi_{S[s]}^{R[s]}, \mathsf{M}, [\mathsf{rch}^{(x)}, \mathsf{ms}_{PQ}])$,

it holds that for any $(\pi_{S[s]}^{R[s]}, \mathsf{M}', [\mathsf{ms}_{PQ}])$ generated by $\mathtt{aReceive}(\mathsf{ltkeys}_{R[s]}, s, \pi_{S[s]}^{R[s]}, \mathsf{C})$, we have $\mathsf{M}' = \mathsf{M}$.

The conditions in Definition 2 allow us to model out-of-order message delivery, and state that if the receiver of stage s gets the message sent by its interlocutor, it will decrypt correctly (implying that both parties were using the same key for encryption). In particular, our notion of correctness implies that the transcripts are *matching* on every stage in which the current receiver actually got the message delivered.

VII. Security model for iMAM protocols

Our security model for iMAM protocols is inspired to the authenticated and confidential channel establishment notion of Jager *et al.* [3]. However, to ease notation and understanding, we follow the *one single* chat approach [1] adopted for messaging protocols, and allow only one protocol instance between any two parties. This simplification does not affect security and it is still possible to derive the generic model from ours. Compared to the security model for Signal [1], that resembles the notion of authenticated key exchange protocol (AKE) ignoring messages and additional data sent during chats, we prove SAID secure in a more realistic model that captures the notions of persistent authentication and confidential channel establishment. To this end, for the unique protocol instance denoted by π_P^Q, we consider the following attribute (in addition to the ones described in the Section VI-B).

Challenge bit $\pi_P^Q.\mathsf{b}[s]$, consisting of a binary value chosen identically and independently at random whenever a new stage $s \in \pi_P^Q$ of an instance is created. The value of a challenge bit is *static* and will be used by the encryption and decryption algorithms (and by the oSend, oReceive, oLoR.AEnc, and oLoR.ADec oracles) in our channel-security game.

Recall that with abuse of notation we write $s \in \pi_P^Q$ for $\pi_P^Q.\mathsf{stages}[s] = 1$, to refer to a stage s that has either happened before in the chat, or is the current stage (of π_P^Q).

A. Adversarial model

We model the adversary as a PPT algorithm $\mathscr{A}$ that can (adaptively) call a series of oracles: oUReg to register users to the system; oCorrupt to corrupt users and learn their long term keys; oStart to initialize a session between two users; oReveal to reveal user's ephemeral keys at a chosen stage; oAccessHSM oracle to simulate momentary-access to a user's device; oSend to simulate the send algorithm; oReceive to simulate the reception algorithm. In the AKE model, the adversary has access to the oTest oracle which returns the message key of a given stage or a random key depending on a challenge bit. In the ACCE model, the adversary has access to the oLoR.AEnc and oRoR.ADec oracles to encrypt and decrypt messages for a given stage. Our oAccessHSM oracle can be seen as a sophisticated twist to the corruption oracle, and follows the notion of trusted environment oracle access given in [21]: we assume that users' long-term secrets are stored within a HSM to which the adversary can only have black-box access.

B. Security Games

Let Π be an iMAM and let $\mathscr{A}$ be a polynomial time algorithm. In this section we define the AKE (resp. ACCE) experiment of $\mathscr{A}$ against Π. We let the adversary $\mathscr{A}$ interact in an adaptive way with all the oracles described in the next paragraph. We consider an additional entity, the challenger $\mathscr{C}$, that runs the aSetup algorithm at the beginning of the experiment, sends mpk to $\mathscr{A}$, and msk to the oracles it administrates. At the end of the experiment $\mathscr{A}$ outputs a tuple $(\pi_P^Q.\mathsf{sid}, s^*, b^*)$.

We consider both the AKE and the ACCE security experiments in this section. We first describe the oracles that are used identically in both experiments, then we define the oracles that are, respectively, only used in the AKE and in the ACCE experiment. As in [1], our AKE model considers a truncated version of the iMAM protocol. More precisely, the users only send the key-exchange material without encrypting or authenticating any message, *i.e.*, they only send the AD when they run the send algorithm. Without this restriction, the adversary may encrypt or decrypt any message of the protocol, and then she may trivially deduce whether a given

key is the real message key or not. On the contrary, our ACCE experiment captures the actual messaging.

$\boxed{\mathsf{oUReg}(P)}$ on input a party P, the user registration oracle runs $\mathtt{aUReg}(P, \mathsf{msk}) \rightarrow (\mathsf{ltkeys}_P, P)$.

$\boxed{\mathsf{oCorrupt}(P)}$ on input a user identity P the corruption oracle checks if P is a registered user, in which case it returns ltkeys_P. Otherwise, it returns $\perp$.

$\boxed{\mathsf{oStart}(P, \mathtt{role}, Q) \rightarrow \pi_P^Q}$ on input a user identity P, a role $\mathtt{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$ and the identity of its intended partner Q, the start-conversation oracle checks if P, Q are registered users, and if not it returns $\perp$. For existing users, it runs $\mathtt{aStart}(\mathsf{ltkeys}_P, \mathtt{role}, Q) \rightarrow \pi_P^Q$ and from then on considers π_P^Q to be an existing chat.

$\boxed{\mathsf{oReveal}(\pi_P^Q.\mathsf{sid}, \mathsf{var}, s)}$ on input a session identifier $\pi_P^Q.\mathsf{sid}$, a variable $\mathsf{var} \in \mathscr{V}$ and a stage $s = (x, y)$ the reveal-state oracle checks if π_P^Q exists, and if $s \in \pi_P^Q$. If both conditions hold, then it returns $\pi_P^Q.\mathsf{var}[s]$.

$\boxed{\mathsf{oAccessHSM}(\pi_P^Q.\mathsf{sid}, \mathsf{fct}, \mathsf{q.input})}$ on input a session identifier $\pi_P^Q.\mathsf{sid}$, some function fct implemented on the trusted module, and a query input $\mathsf{q.input}$, the access-trusted-module oracle returns the result of the black-box on the given inputs $\mathsf{fct}(\mathsf{q.input})$.

Oracles for the AKE model

$\boxed{\mathsf{oTest}(\pi_P^Q.\mathsf{sid}, s)}$ on input a session identifier $\pi_P^Q.\mathsf{sid}$ and a stage s, if:

- π_P^Q does not exist, or
- $s \notin \pi_P^Q$, or
- $\pi_P^Q.\mathsf{k}[s] = \perp$, or
- this oracle has already been called on $(\pi_P^Q.\mathsf{sid}, s)$ or $(\pi_Q^P.\mathsf{sid}, s)$

then the test oracle returns $\perp$. Otherwise, if $\pi_P^Q.\mathsf{b}[s] = 0$, it returns a randomly generated key $\mathsf{k} \xleftarrow{\$} \mathscr{K}$, else it returns the actual message key $\pi_P^Q.\mathsf{k}[s]$ (which corresponds to what is generally called the session key in the AKE model).

$\boxed{\mathsf{oSend}(\pi_P^Q.\mathsf{sid}, s, \mathsf{AD})}$ on input a session identifier $\pi_P^Q.\mathsf{sid}$, a stage $s = (x, y)$, and a (possibly empty) string of additional data AD, this sending oracle checks that:

1) π_P^Q exists,
2) $s \notin \pi_P^Q$,
3) P is the sender for stage s,
4) there exists $s' \in \pi_P^Q$ such that $s \in \mathsf{next}(s')$,

then if one of these conditions does not hold, the oracle returns $\perp$. Otherwise, if AD is not empty, it simulates the sending algorithm (*i.e.*, it updates π_P^Q as $\mathtt{aSend}$ would) using AD as additional data, which it then returns; alternatively, it generates $\mathtt{aRKGen}(1^\lambda) \rightarrow (\mathsf{rchsk}_P^{(x)}, \mathsf{rchpk}_P^{(x)})$, then runs $\mathtt{aSend}(\mathsf{ltkeys}_P, s, \pi_P^Q, \perp, (\mathsf{rchsk}_P^{(x)}, \mathsf{rchpk}_P^{(x)}), \mathsf{ms}_{PQ})$ and returns the additional data. Without loss of generality, we suppose that the AD given by the attacker can always be parsed into something correct, even if that means truncating or padding it. Note that only the sender's transcript is updated when this oracle is called, under the condition it had not been called at that stage before, *i.e.*, its usage is restricted to once per stage. If AD is empty, this oracle sets $\mathsf{SHU}(\pi_P^Q, s) = 1$ to denote that it has been *sent like an honest user* on stage s, else it sets $\mathsf{SHU}(\pi_P^Q, s) = 0$.

$\boxed{\mathsf{oReceive}(\pi_P^Q.\mathsf{sid}, s, \mathsf{AD})}$ on input a session identifier $\pi_P^Q.\mathsf{sid}$, a stage $s = (x, y)$, and a string of additional data AD, this receiving oracle checks condition 1 from oSend along with:

5) P is the $\mathsf{receiver}$ for stage s,
6) if $x \neq 0$, there exists $s' = (x', y') \in \pi_P^Q.\mathsf{stages}$ such that $x' = x - 1$,

then if one of these conditions does not hold, the oracle returns $\perp$. This oracle then runs $\mathtt{aReceive}$ with AD. As it was for oSend, the AD is "automatically" assumed to be correct. Note that only the $\mathsf{receiver}$'s transcript is updated by this oracle, allowing out-of-order messages or derailing, in case the AD is different from the one used in the sending oracle for that level. If $\pi_P^Q.Tr[s] = \mathsf{AD}$ and $\mathsf{SHU}(\pi_Q^P, s) = 1$, then this oracle sets $\mathsf{RHU}(\pi_Q^P, s) = 1$ to denote that it has been *received like an honest user* on stage s, else $\mathsf{RHU}(\pi_Q^P, s) = 0$

Oracles for the ACCE model

$\boxed{\mathsf{oSend}(\pi_P^Q.\mathsf{sid}, s, \mathsf{M}_0, \mathsf{M}_1, \mathsf{AD}, \mathsf{C})}$ is very similar to oSend for AKE, with the additional input of two messages $\mathsf{M}_0, \mathsf{M}_1$, and a ciphertext C (empty if AD is), checking conditions 1-4 to know if it should return $\perp$. If the conditions hold, if AD and C are not empty, it simulates the sending algorithm using AD as additional data and $\mathsf{M}_{\pi_P^Q.\mathsf{b}[s]}$ as the message (*i.e.*, it updates π_P^Q as $\mathtt{aSend}$ would) and returns C; alternatively, it generates $\mathtt{aRKGen}(1^\lambda) \rightarrow \mathsf{rch}^{(x)} = (\mathsf{rchsk}^{(x)}, \mathsf{rchpk}^{(x)})$, then runs $\mathtt{aSend}(\mathsf{ltkeys}_P, s, \pi_P^Q, \mathsf{M}_{\pi_P^Q.\mathsf{b}[s]}, \mathsf{rch}^{(x)}, \mathsf{ms}_{PQ})$ to obtain a ciphertext C', and return it. As it is in AKE, if AD and C are empty, this oracle sets $\mathsf{SHU}(\pi_P^Q, s) = 1$, else it sets $\mathsf{SHU}(\pi_P^Q, s) = 0$. Finally, this oracle adds C or C' to a list $\mathscr{L}_{P,Q,s}$.

$\boxed{\mathsf{oLoR.AEnc}(\pi_P^Q.\mathsf{sid}, s, \mathsf{M}_0, \mathsf{M}_1, \mathsf{AD})}$ on input a session identifier $\pi_P^Q.\mathsf{sid}$, a stage $s = (x, y)$, two messages $\mathsf{M}_0, \mathsf{M}_1$, and a (possibly empty) string of additional data AD, the encryption oracle checks condition 1, and that $s \in \pi_P^Q$, then encrypts both messages using either AD or some additional data generated by $\mathtt{aRKGen}$, then if either encryption is $\perp$, it returns $\perp$, otherwise it returns the result C of the encryption of $\mathsf{M}_{\pi_P^Q.\mathsf{b}[s]}$. Note that this oracle is meant to be for the adversary to play with, no message is ever sent or received. Finally, this oracle adds C to a list $\mathscr{L}_{P,Q,s}$.

$\boxed{\mathsf{oReceive}(\pi_P^Q.\mathsf{sid}, s, \mathsf{C}, \mathsf{AD})}$ this receiving oracle is almost identical to the one for AKE, with the additional input of a ciphertext C. It checks conditions 1-2 and 5-6, and returns $\perp$ if they do not hold. If they hold, then this oracle runs $\mathtt{aReceive}$ for C, which returns the message M. If

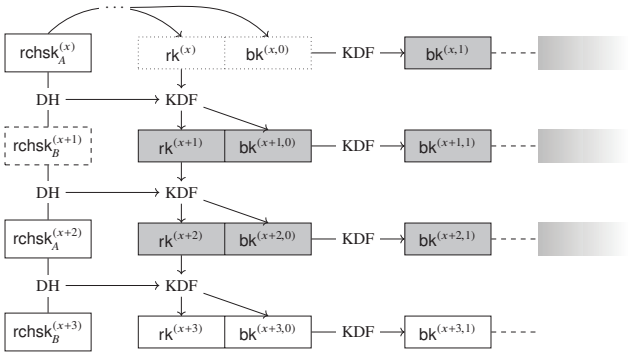

Fig. 3: An illustration of which keys are computable (in grey) if the adversary reveals stage $(x, 0)$ (dotted) and injects the ratchet key of level $x + 1$ (dashed).

$\mathsf{C} \in \mathscr{L}_{P,Q,s} \cup \mathscr{L}_{Q,P,s}$ or $\pi_P^Q.\mathsf{b}[s] = 0$, this oracle returns $\perp$, else it returns M. If $\pi_P^Q.Tr[s] = \mathsf{AD}$ and $\mathsf{SHU}(\pi_Q^P, s) = 1$, then this oracle sets $\mathsf{RHU}(\pi_Q^P, s) = 1$, else $\mathsf{RHU}(\pi_Q^P, s) = 0$.

$\mathsf{oRoR.ADec}(\pi_P^Q.\mathsf{sid}, s, \mathsf{C}, \mathsf{AD})$ on input a session identifier $\pi_P^Q.\mathsf{sid}$, a stage $s = (x, y)$, a ciphertext C, and a string of additional data AD, the decryption oracle checks condition 1, and that $s \in \pi_P^Q$, then if $\mathsf{C} \in \mathscr{L}_{P,Q,s} \cup \mathscr{L}_{Q,P,s}$ or $\pi_P^Q.\mathsf{b}[s] = 0$, this oracle returns $\perp$ [10], else it decrypts the ciphertext and returns the result to the adversary.

Winning conditions of SAID. We recall that at the end of the experiment, the adversary outputs $(\pi_P^Q.\mathsf{sid}, s^*, b^*)$.

We will now list the conditions under which an adversary trivially knows or computes a message key $\pi_P^Q.\mathsf{k}^{(s)}$ for a stage s depending on which oracles were called. Unless specified otherwise, the calls are made on P's side. We choose to separate the oracle calls into two categories: those that induce the compromise of the long-term shared key ms_{PQ}, weak or full (in this case the security of SAID becomes equivalent to the security of Signal for previous stages), and the others (the adversary cannot compute past keys by herself).

Since we allow the adversary to "derail" the users by giving them a different ratchet key at the same level, *i.e.*, giving a different AD to oSend and $\mathsf{oReceive}$, it is possible that $\pi_P^Q.\mathsf{k}^{(s)} \neq \pi_Q^P.\mathsf{k}^{(s)}$. This is verifiable formally by checking the transcripts on both sides. However, since the challenge is based on a key for a specific user, this means that it has to be taken into account in the winning conditions.

Similarly, the adversary could derail the users right from the beginning of the conversation, if she intercepts all of the initiator's messages, and calls $\mathsf{oAccessHSM}$ on the routine $\mathtt{IBS.Sign}(\mathtt{IBS.mpk}, \mathtt{IBS.sk}_P, \cdot)$ in order to obtain a valid signature from the initiator on a value of her choice, which is then sent to the responder, she will derail the conversation and know the (fake) master secret, but she has to take the initiator's place and thus trivially knows every key on the receiver's side.

[10]The decryption returns $\perp$ if the bit is 0, as in decription oracle of the LH-AEAD security model.

The initiator's side, however, is still secure – note that since the responder never replied, there is only one level.

The adversary could make SAID fall back in Signal's security if the master secret is compromised, *i.e.*, is known and/or used, which happens if:

1) she impersonates the initiator via the attack described above, or
2) she called $\mathsf{oCorrupt}$ on the session's responder R – and thus knows idsk_R, which gives the master secret $e(\mathsf{idsk}_R, h)$ –, or
3) she simply called $\mathsf{oAccessHSM}$ on the routine $\mathsf{KDF}_*(\mathsf{ms}_{PQ}, \cdot)$ at stage $s_A = (x_A, y_A)$, thus potentially leaking any previous stage.

Note that cases 1 and 2 are full compromises, when case 3 is a weak compromise. If we are in this situation, then the adversary trivially knows the message key $\pi_P^Q.\mathsf{k}^{(s)}$ of stage $s = (x, y)$ anterior to s_A, if she found herself in either of the following conditions:

- $x = 0$, *i.e.*, she is on the first level[11],
- she called $\mathsf{oReveal}$ with $\mathsf{var} = \mathsf{bk}$ on a stage $(x, y' < y)$, or with $\mathsf{var} = \mathsf{k}$ on s, for P, or for Q if $\pi_P^Q.\mathsf{rchpk}[s] = \pi_Q^P.\mathsf{rchpk}[s]$,
- she called $\mathsf{oReveal}$ with $\mathsf{var} = \mathsf{rchsk}$ *and* with $\mathsf{var} = \mathsf{rk}$ on a stage $(x-1, y')$ for the sender of that stage, under the condition, if that sender is Q, that $\pi_P^Q.\mathsf{rchpk}[(x-1, y')] = \pi_Q^P.\mathsf{rchpk}[(x-1, y')]$,
- she injected the ratchet key of level $x - 1$, *i.e.*,
  - if P is the current sender (resp. receiver), she called oSend (resp. $\mathsf{oReceive}$) with her own AD for a stage $(x-1, 0)$ (resp. $(x-1, \pi_P^Q.\mathsf{rec}[s])$), so either $\mathsf{SHU}(\pi_P^Q, (x-1, 0))$ or $\mathsf{RHU}(\pi_P^Q, (x-1, \pi_P^Q.\mathsf{rec}[s]))$ is 0,

  and knew the root key of that level, meaning she either
  - called $\mathsf{oReveal}$ with $\mathsf{var} = \mathsf{rk}$ on a stage $(x-2, y'')$, for P – or for Q, under the condition that $\pi_P^Q.\mathsf{rchpk}[(x-2, y'')] = \pi_Q^P.\mathsf{rchpk}[(x-2, y'')]$ – or
  - actually also injected the ratchet key of level $x - 2$, while knowing the root key of that level, etc.,

  as illustrated in Figure 3.

Moreover, if the adversary continuously injects ratchet keys, starting at level $x_I \leq x_A$, then until two honest ratchet keys in a row are generated, say, at levels x_H and $x_H + 1$, all message keys from level x_I up to, and including, level x_H are known to the adversary[12]. This is a straightforward extension to the attack presented in Figure 3.

Now if neither $\mathsf{oCorrupt}$ nor $\mathsf{oAccessHSM}$ were called, or if the adversary called $\mathsf{oAccessHSM}$ on stage s_A, such that $s_A < s$, and

$$\mathsf{RHU}(\pi_P^Q, (x_H, \pi_P^Q.\mathsf{rec}[s])) = 1 \text{ or } \mathsf{SHU}(\pi_P^Q, s) = 1,$$

then the only way for the adversary to trivially know the message key $\pi_P^Q.\mathsf{k}^{(s)}$ for a stage $s = (x, y)$ (posterior to s_H, if

[11]All keys of stages $(0, y)$ can be derived from the master secret.

[12]In the weak compromise case, she still has to call $\mathsf{oAccessHSM}$.

relevant) is if she called oReveal with var = k on the stage s for P, or Q if $\pi_P^Q.\mathsf{rchpk}[s] = \pi_Q^P.\mathsf{rchpk}[s]$.

Note that because the adversary no longer uses oAccessHSM, and did not call oCorrupt, an oReveal query can only compromise a single stage, even with var = bk, because the master secret is not known to her, yet it is always needed for the key derivations.

Finally, in both the AKE and the ACCE experiment, if:

- π_P^Q exists, and
- $s^* \in \pi_P^Q$, and
- none of the conditions mentioned in this paragraph hold,

then if $b^* = \pi_P^Q.\mathsf{b}[s]$, the challenger returns 1, else it returns 0. Otherwise, the challenger picks a random bit $b' \xleftarrow{\$} \{0,1\}$ and returns it.

Definition 3 (AKE/ACCE-security): Let Π be an iMAM. Π is said to be AKE/ACCE secure if for any polynomial time adversary $\mathscr{A}$, the probability that $\mathscr{A}$ wins the AKE/ACCE experiment is negligibly close to $1/2$.

C. Security Proofs

In this section, we show that SAID is secure in both the AKE and the ACCE model.

Theorem 1: If IBS is EUFCMA-secure, then SAID is AKE-secure under the BCDH and the CDH assumptions in the random oracle model.

Theorem 2: If SAID is AKE-secure and AEAD is LH-AEAD-secure then SAID is ACCE secure.

We give the proof sketches of these two theorems. Our proofs use the sequences of games approach introduced by Shoup in [22]. The complete proofs are given in the full version[13].

Proof (sketch) of Theorem 1:

We partition the analysis for two individual cases:

- **Case 1**: (i) The adversary calls the oAccessHSM oracle for the conversation π_P^Q after the stage s_* is generated or (ii) the adversary calls the oCorrupt oracle on the user P or Q (in this case, the idea is that the key ms_{PQ} is *corrupted*, hence the security of SAID is equivalent to the one of Signal).
- **Case 2**: (i) The adversary does not call the oAccessHSM oracle for the conversation π_P^Q after the stage s_* is generated and (ii) the adversary does not call the oCorrupt oracle on users P and Q (in this case, the idea is that the key ms_{PQ} is not *corrupted*, hence the adversary must guess ms_{PQ} to win).

Case 1: We observe that if the stage $s_* = (x_*, y_*)$ satisfies the winning conditions, then there exists $x_{**} \leq x_*$ such that:

- $\mathsf{RHU}(\pi_P^Q, (x_{**}, \pi_P^Q.\mathsf{rec}[s_*]))$ or $\mathsf{SHU}(\pi_P^Q, (x_{**}, 0))$ is 1,
- $\mathsf{RHU}(\pi_P^Q, (x_{**} - 1, \pi_P^Q.\mathsf{rec}[s_*]))$ or $\mathsf{SHU}(\pi_P^Q, (x_{**} - 1, 0))$ is 1,
- the adversary never queries $(\pi_P^Q.\mathsf{sid}, \mathsf{var}, (x, y))$ to the oReveal oracle such that $\mathsf{var} \in \{\mathsf{bk}, \mathsf{rk}\}$ (resp. $\{\mathsf{k}, \mathsf{rchsk}\}$), and $x_{**} \leq x \leq x_*$ and $0 \leq y \leq y_*$ (resp. $(x, y) = (x_*, y_*)$) during the experiment.

We first show that the probability that the adversary sends a query $(\mathsf{ms}_{PQ}^*, \Delta^*, \mathsf{rk}^*)$ to the random oracle that simulates KDF_1 such that $\Delta^* = \Delta^{x_{**}}$ is negligible. We prove this claim by reduction. Assume that there exists a polynomial time algorithm $\mathscr{A}$ for whom this probability is non negligible. We show how to build an algorithm $\mathscr{B}$ that breaks the CDH experiment. The idea is that if the adversary has a non-negligible advantage to win the AKE experiment, then she sends $\Delta^{x_{**}}$ to the random oracle that simulates Let q_{H} be the number of queries sent to the random oracles, which is the result of a CDH instance.

We then show that the probability that the adversary sends a query $(\mathsf{ms}_{PQ}^*, \Delta^*, \mathsf{rk}^*)$ or $(\mathsf{ms}_{PQ}^*, \mathsf{bk}^*, \mathsf{tag}^*)$ to the random oracles that simulates KDF_1 and the function HKDF used in KDF_2, such that $\mathsf{rk}^* \in \{\mathsf{rk}^{x_{**}}, \mathsf{rk}^{x_{**}+1}, \ldots, \mathsf{rk}^{x_*}\}$, or such that $\mathsf{bk}^* \in \{\mathsf{bk}^{(x_*,0)}, \mathsf{bk}^{(x_*,1)}, \ldots, \mathsf{bk}^{(x_*,y_*)}\}$ is negligible. The idea is that if the adversary does not know $\Delta^{x_{**}}$, then she is not able to guess all values that are generated from the hash of $\Delta^{x_{**}}$.

Finally, if $\mathscr{A}$ never sends $\mathsf{bk}^{(x_*,y_*)}$ to the random oracle that simulates the KDF function, then the key k^{s_*} is indistinguishable from random. In this case, the probability that $\mathscr{A}$ wins the game is exactly $1/2$.

Case 2: We observe that if the stage $s_* = (x_*, y_*)$ satisfies the winning condition, then there exists $s_{**} < s_*$ such that the oracle oAccessHSM is not called after the stage s_{**} is generated and $\mathsf{SHU}(\pi_P^Q, s) = 1$ or $\mathsf{RHU}(\pi_P^Q, s) = 1$. We want to remove the cases where the adversary guesses the random tag tag^{s_*} before it is generated, and sends it to the oracle oAccessHSM. In this case the adversary should guess the key k^{s_*} without breaking the winning conditions. More formally, we show that the probability that the adversary has sent the query $q = (\mathsf{bk}^{s_*}, \mathsf{tag}^{s_*})$ to the oracle oAccessHSM such that $\mathsf{tag}^* = \mathsf{tag}^{s_*}$ before the stage s_* is generated (hence before tag^{x_*} is generated) is at most the probability that it guesses tag^{x_*} at random in q_{HSM} tries, where q_{HSM} is the numer of queries sent to oAccessHSM, which is negligible. The idea is that the adversary must guess the value at random since it does not exists when she calls the oAccessHSM oracle.

Next, we show that the probability that the adversary chooses the additional data $\mathsf{AD} = (P, Q, \mathsf{rchpk}_P^{(0)}, \pi_P^Q.\mathsf{rec}[(0,0)]), h, \mathsf{sgn})$ by herself when she calls the oSend oracle on the stage $(0, \pi_P^Q.\mathsf{rec}[s]))$ and π_Q^P such that $\mathtt{IBS.Vrfy}(\mathsf{IBS.param}, I, (I, R, \mathsf{rchpk}_I^{(0)}, h), \mathsf{sgn}) = 1$ is negligible. We prove this claim by reduction. Assume that there exists a polynomial time algorithm $\mathscr{A}$ such this probability is non-negligible. We show how to build a polynomial time algorithm $\mathscr{B}$ that breaks the unforgeability of the identity based signature.

Finally, we show that the probability that the adversary sends a query that contains ms_{PQ} to the random oracles is negligible. We prove this claim by reduction. Assume that there exists a polynomial time algorithm $\mathscr{A}$ such that this

[13] ia.cr/2019/367

probability is non negligible. We show how to build an algorithm $\mathscr{B}$ that breaks the BCDH assumption. The idea is that to guess ms_{PQ}, the adversary must compute the bilinear computational Diffie-Hellman from $\mathtt{H}(Q)$, $\mathsf{ID.mpk}$, and h. We recall that since the adversary does not generate the signature of h, this value was honestly chosen by the challenger.

On the other hand, if $\mathscr{A}$ never sends ms_{PQ} to the random oracle that simulates the KDF functions and $\mathscr{A}$ never sends $\mathsf{bk}^{(x_*,y_*)}$ to the $\mathsf{oAccessHSM}$ oracle on routine $\mathsf{R.KDF}_*(\mathsf{ms}_{PQ},\cdot)$, then the key k^{s_*} is indistinguishable from a random value. In this case, the probability that $\mathscr{A}$ wins the game is exactly $1/2$.

Finally, in any case, the probability that the adversary wins the game is negligibly close to $1/2$. □

proof (sketch) Theorem 2:

We prove this theorem by reduction. Assume that AEAD is LH-AEAD-secure, and assume that there exists an algorithm $\mathscr{A}$ that breaks the ACCE security of SAID. We show how to build an algorithm $\mathscr{B}$ that breaks the AKE security of SAID. The algorithm $\mathscr{B}$ simulates the ACCE experiment to $\mathscr{A}$ using the keys produced by the test oracle. If $\pi_P^Q.\mathsf{b}[s] = 0$, then the message key is randomly chosen. In this case, encryptions and decryptions are independent from the keys exchanged in SAID, hence, winning the ACCE experiment is equivalent to breaking the security of the AEAD scheme, so the advantage of $\mathscr{A}$ is negligible. On the other hand, if $\pi_P^Q.\mathsf{b}[s] = 1$ then $\mathscr{B}$ encrypts and decrypts using the real message key. In this case, $\mathscr{A}$ wins with a non-negligible advantage by hypothesis. Finally, depending on the response of $\mathscr{A}$, the adversary guesses the bit b with non-negligible probability. □

ACCE Security of Signal. We note that the security of Signal in our model can be proven in a similar way as for SAID. We recall that our model is stronger than the one proposed in [1] since it allows the adversary to inject her own ratchet keys in the additional data, and considers the un-truncated version of the Signal protocol.

VIII. Conclusion

In this paper, we studied the Signal messaging protocol from a new angle and proposed a variant, SAID, that we proved secure in the authenticated and confidential channel establishment (ACCE) model.

We focused on fixing several weaknesses in Signal to build SAID, which inherently led us to a more secure protocol. Our construction stays very close to Signal, with some additional or differently computed keys here and there.

Cohn-Gordon *et al.* [1] proved the security of Signal in the authenticated key exchange (AKE) model, which required them to consider the protocol with several modifications. We give in this paper a proof of SAID in the ACCE model, in order to be able to fully consider the protocol.

We generalized SAID by defining Identity-Based Multi-Stage Asynchronous Messaging Protocols, for which we gave the security model that we used in our proof.

In the future, we would like to implement our solution to prove its practical efficiency. Moreover, several works recently focused on group messaging, and the *Messaging Layer Security IETF Working Group* was created in early 2018. Following this, we would like to see how to extend SAID to a multi-user setting.

Acknowledgments

This work was supported in part by the French ANR through grants 16 CE39 0012 (SafeTLS), 16 CE39 0004 (IDFIX) and the e-COST Action IC 1403.

References

[1] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the signal messaging protocol," in *Proceedings of Euro S&P*. IEEE, 2017.

[2] K. Cohn-Gordon, C. Cremers, and L. Garratt, "On post-compromise security," in *Proceedings of CSF*. IEEE, 2017.

[3] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of TLS-DHE in the standard model," in *Proceedings of CRYPTO*. Springer, 2012.

[4] J. Jaeger and I. Stepanovs, "Optimal channel security against fine-grained state compromise: The safety of messaging," in *CRYPTO*, 2018.

[5] K. Cohn-Gordon, C. Cremers, L. Garratt, J. Millican, and K. Milner, "On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees," in *Proceedings of CCS*. ACM, 2018.

[6] M. Bellare, A. Camper Singh, J. Jaeger, M. Nyayapati, and I. Stepanovs, "Ratcheted encryption and key exchange: The security of messaging," in *Proceedings of CRYPTO*. Springer, 2017.

[7] B. Poettering and P. Rösler, "Ratcheted key exchange, revisited," IACR ePrint archive, report 296/2018, 2018.

[8] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Proceedings of CRYPTO*. Springer, 1993.

[9] M. Fischlin and F. Günther, "Multi-stage key exchange and the case of google's QUIC protocol," in *Proceedings of CCS*. ACM, 2014.

[10] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proceedings of CRYPTO*. Springer, 1985.

[11] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *Proceedings of CRYPTO*. Springer, 2001.

[12] M. Bellare, C. Namprempre, and G. Neven, "Security proofs for identity-based identification and signature schemes," *Journal of Cryptology*, 2009.

[13] J. Jiang, C. He, and L.-g. Jiang, "On the design of provably secure identity-based authentication and key exchange protocol for heterogeneous wireless access," in *NMC*. Springer, 2005.

[14] P. Rogaway, "Authenticated-encryption with associated-data," in *Proceedings of CCS*. ACM, 2002.

[15] K. G. Paterson, T. Ristenpart, and T. Shrimpton, "Tag size does matter: Attacks and proofs for the tls record protocol," in *Proceedings of ASIACRYPT*. Springer, 2011.

[16] F. Bao, R. H. Deng, and H. Zhu, "Variations of diffie-hellman problem," in *ICICS*. Springer, 2003.

[17] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *TrustCom/BigDataSE/ISPA*. IEEE, 2015.

[18] M. Scheir, J. Balasch, A. Rial, B. Preneel, and I. Verbauwhede, "Anonymous split e-cash - toward mobile anonymous payments," *ACM Trans. Embedded Comput. Syst.*, 2015.

[19] C. Schnorr, "Efficient identification and signatures for smart cards," in *Proceedings of CRYPTO 1989*, 1989, pp. 239–252.

[20] S. Chatterjee, A. Menezes, and F. Rodríguez-Henríquez, "On instantiating pairing-based protocols with elliptic curves of embedding degree one," *IEEE Trans. Computers*, 2017.

[21] D. Boneh and V. Shoup, *A graduate course in applied cryptography*. Online edition, https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_4.pdf, 2017.

[22] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs," Cryptology ePrint Archive, Report 2004/332, 2004, https://eprint.iacr.org/2004/332.