2017 IEEE European Symposium on Security and Privacy

A Formal Security Analysis of the Signal Messaging Protocol

(Extended abstract: full version available at [10])

Katriel Cohn-Gordon*, Cas Cremers*, Benjamin Dowling†, Luke Garratt*, Douglas Stebila‡

*University of Oxford, UK
katriel.cohn-gordon@cs.ox.ac.uk
cas.cremers@cs.ox.ac.uk
luke.garratt@cs.ox.ac.uk

†Royal Holloway, University of London, UK
benjamin.dowling@rhul.ac.uk

‡McMaster University, Canada
stebilad@mcmaster.ca

***Abstract*—Signal is a new security protocol and accompanying app that provides end-to-end encryption for instant messaging. The core protocol has recently been adopted by WhatsApp, Facebook Messenger, and Google Allo among many others; the first two of these have at least 1 billion active users. Signal includes several uncommon security properties (such as "future secrecy" or "post-compromise security"), enabled by a novel technique called *ratcheting* in which session keys are updated with every message sent. Despite its importance and novelty, there has been little to no academic analysis of the Signal protocol.**

We conduct the first security analysis of Signal's Key Agreement and Double Ratchet as a multi-stage key exchange protocol. We extract from the implementation a formal description of the abstract protocol, and define a security model which can capture the "ratcheting" key update structure. We then prove the security of Signal's core in our model, demonstrating several standard security properties. We have found no major flaws in the design, and hope that our presentation and results can serve as a starting point for other analyses of this widely adopted protocol.

1. Introduction

Revelations about mass surveillance of communications have made consumers more privacy-aware. In response, scientists and developers have proposed techniques which can provide security for end users even if they do not fully trust the service providers. For example, the popular messaging service WhatsApp was unable to comply with Brazilian government demands for users' plaintext messages [7] because of its end-to-end encryption.

Early instant messaging systems did not provide much security. While some systems did encrypt traffic between the user and the service provider, the service provider retained the ability to read the plaintext of users' messages. Off-the-Record Messaging [8, 17] was one of the first security protocols for instant messaging: acting as a plugin to a variety of instant messaging applications, users could authenticate each other using public keys or a shared secret passphrase, and obtain end-to-end confidentiality and integrity. One novel feature of OTR was its fine-grained key freshness: along with each message round trip, users established a fresh ephemeral Diffie–Hellman (DH) shared secret. Since it was not possible to work backward from a later state to an earlier state and decrypt past messages, this technique became known as *ratcheting*; in particular, *asymmetric* ratcheting since it involves asymmetric (public key) cryptography. OTR saw relatively limited adoption, but its ratcheting technique can be seen in modern security protocols.

Perhaps the first secure instant message protocol to achieve widespread adoption was Apple's iMessage, a proprietary protocol that provides end-to-end encryption. A notable characteristic of iMessage is that it automatically manages the distribution of users' long-term keys, and in particular (as of this writing) users have no interface for verifying friends' keys. iMessage, unfortunately, has a variety of flaws that seriously undermine its security [23].

The Signal Protocol. While there has been a range of activity in end-to-end encryption for instant messaging [19, 48], the most prominent recent development in this space has been the Signal messaging protocol, "a ratcheting forward secrecy protocol that works in synchronous and asynchronous messaging environments" [36, 37]. Signal's goals include end-to-end encryption as well as advanced security properties such as perfect forward secrecy and "future secrecy".

The Signal protocol, and in particular its ratcheting construction, has a relatively complex history. TextSecure [37] was a secure messaging app and the predecessor to Signal. It contained the first definition of Signal's "Double Ratchet", which effectively combines ideas from OTR's asymmetric ratchet and a *symmetric* ratchet (which applies a symmetric key derivation function to create a new key, but does not incorporate fresh DH material, similar to so-called "forward-secure" symmetric encryption [4]). TextSecure's combined ratchet was referred to as the "Axolotl Ratchet", though the name Axolotl was used by some to refer to the entire protocol. TextSecure

K.C-G. thanks Merton College and the Oxford CDT in Cyber Security for their support.
D.S. was supported in part by Australian Research Council (ARC) Discovery Project grant DP130104304, Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grant RGPIN-2016-05146 and Discovery Accelerator Supplement RGPAS 492986-2016.

DOI 10.1109/EuroSP.2017.27

was later merged with RedPhone, a secure telephony app, and was renamed Signal[1], the name of both the instant messaging app and the cryptographic protocol. In the rest of this paper, we will be discussing the cryptographic protocol only.

The Signal cryptographic protocol has seen explosive uptake of encryption in personal communications: it (or a variant) is now used by Google Allo [38], WhatsApp [50], Facebook Messenger [20], as well as a host of variants in "secure messaging" apps, including Silent Circle [40], Pond [34], and (via the OMEMO extension [47] to XMPP) Cryptocat v2 [27], Conversations [12], and ChatSecure [2].

Security of Signal. One might expect this widespread uptake of the Signal protocol to be accompanied by an in-depth security analysis and examination of the design rationale, in order to: (i) understand and specify the security assurances which Signal is intended to provide; and (ii) verify that it provides them.

Surprisingly, this is not yet the case. There currently is little documentation available on the current version of the Signal protocol, and no in-depth security analysis, although the developers have recently started work on some specifications for various components of the protocol. This is in stark contrast to the ongoing development of the next version of the Transport Layer Security protocol, TLS 1.3, which explicitly involves academic analysis in its development [6, 14, 18, 26, 29, 35].

Frosch et al. [21, 22] performed a security analysis of TextSecure v3, showing that in their model the computation of the long-term symmetric key which seeds the ratchet is a secure one-round key exchange protocol, and that the key derivation function and authenticated encryption scheme used in TextSecure are secure. However, it did not cover any of the security properties of the ratcheting mechanisms.

In addition, Frosch et al. identified an unknown key share (UKS) attack against TextSecure, because the cryptographic material was not bound to the identities. This attack is also not prevented by the core of the Signal protocol, but can be prevented at higher layers. We therefore explicitly exclude the UKS attack from our security analysis. For details, see the full version of this paper.

Providing a security analysis for the Signal protocol is challenging for several reasons. First, Signal employs a novel and unstudied design, involving over ten different types of keys and a complex update process which leads to various "chains" of related keys. It therefore does not directly fit into existing analysis models. Second, some of its claimed properties have only recently been formalised [11]. Finally, as a more mundane obstacle, the protocol is not substantially documented beyond its source code.

1. TextSecure v1 was based on OTR; in v2 it migrated to the Axolotl Ratchet and in v3 made some changes to the cryptographic primitives and the wire protocol. Signal is based on TextSecure v3.

1.1. Contributions

We provide the first in-depth formal security analysis of the cryptographic core of the Signal messaging protocol, which is used by more than a billion users.

To achieve this, we develop a multi-stage key exchange security model with adversarial queries and freshness conditions that capture the security properties intended by Signal. Compared to previous multi-stage key exchange models which involve a single sequence of stages within each session, our model considers a *tree* of stages to model the various "chains" in Signal. Our security model characterizes many detailed security properties of Signal, providing the first formal definition of Signal's security goals. Among the interesting aspects of our model are the subtle differences between security properties of keys derived via symmetric and asymmetric ratcheting.

We subsequently prove that the cryptographic core of Signal is secure in our model, providing the first formal security guarantees for Signal. We give a theorem statement in this paper; the complete proof is included in the full version [10].

In practice, Signal is more than just its key exchange protocol. In Section 6, we describe many other aspects of Signal that are not covered by our analysis, which we believe are a rich opportunity for future research. We hope our presentation of the protocol in Section 2 can serve as a starting point for understanding Signal's core.

1.2. Additional Related Work

Symmetric ratcheting and DH updates (asymmetric ratcheting) are not the only way of updating state to ensure forward secrecy—i.e., that compromise of current state cannot be used to decrypt past communications. Forward-secure public key encryption [9] allows users to publish a short unchanging public key; messages are encrypted with knowledge of a time period, and after receiving a message, a user can update their secret key to prevent decryption of messages from earlier time periods.

Signal's asymmetric ratcheting, which it inherits from the design of OTR [8], have been claimed to offer properties such as "future secrecy". Future secrecy of protocols like Signal has been discussed in depth by Cohn-Gordon, Cremers, and Garratt [11]. Their key observation is that Signal's future secrecy is (informally) specified with respect to a passive adversary, and therefore turns out to be implied by the formal notion of forward secrecy. Instead, they observe that mechanisms such as asymmetric ratcheting can be used to achieve a substantially stronger property against an active adversary. They formally define this property as "post-compromise security", and show how this substantially raises the bar for resourceful network attackers to attack specific sessions. Furthermore, their analysis indicates that post-compromise security may hold of Signal depending on subtle details related to device state reset and the handling of multiple devices.

In concurrent work released after the initial version of this paper, Bellare et al. [3] develop security

definitions for ratcheted key exchange in a more general context than the Signal protocol, and describe a Diffie–Hellman based protocol that is somewhat similar to the Signal protocol. Also concurrently and published at EuroS&P 2017, Kobeissi et al. [28] use ProVerif and CryptoVerif to analyze an implementation of Signal in a JavaScript variant called ProScript.

Recently, Green and Miers [24] suggest using puncturable encryption to achieve fine-grained forward security with unchanging public keys: instead of deleting or ratcheting the secret key, it is possible to modify it so that it cannot be used to decrypt a certain message. While this is an interesting approach (especially for its relative conceptual simplicity), we focus on Signal due to its widespread adoption.

Overview. In Section 2 we give a detailed presentation of the Signal protocol. We follow this by a high-level description of its threat model in Section 3, and a formal security model in Section 4. In Section 5 we prove security of Signal's core in our model. As a first analysis of a complex protocol our model has some limitations and simplifying assumptions, discussed in detail in Section 6. We conclude in Section 7.

2. The Core Signal Protocol

Basic setup. The Signal protocol aims to send encrypted messages from one party to another. It assumes each party has a long-term public/private key pair, referred to as the identity key. However, since the parties might be offline at any point in time, standard authenticated key-exchange (AKE) solutions cannot be directly applied. For instance, using DH key-exchange to achieve perfect-forward secrecy requires both parties to contribute new ephemeral DH keys, but the recipient may be offline at the time of sending.

Instead, Signal implements an asynchronous transmission protocol, by requiring potential recipients to pre-send batches of ephemeral public keys. When the sender wishes to send a message, she obtains keys for the recipient from an intermediate server (which only acts as a buffer), and performs an AKE-like protocol using the long-term and ephemeral keys to compute a message encryption key.

This basic setup is then extended by making the message keys dependent on all previously performed exchanges between the parties, using a combination of "ratcheting" mechanisms to form "chains". New random and secret values are also introduced into the computations at various points, influencing future message keys computed by the communicating partners.

Motivation and Scope. The Signal protocol uses an intricate design whose rationale is currently not formally documented, although there has been substantial informal discussion on mailing lists and blog posts. Our focus is to study the existing protocol: we aim simply to report what Signal *is*, not why any of its design choices were made.

It is not entirely straightforward to pin down a precise definition of the intended usage and security properties of Signal. Our descriptions in this section were aided by some documentation but the ultimate authority was the implementation[2] [36]. After the time of writing, Open Whisper Systems published high-level specifications for X3DH [45] and the Double Ratchet [44] which help to clarify many details, although the codebase is still necessary to obtain a full definition and the specification does not contain detailed definitions of the security goals.

2.1. Protocol Overview

A party using Signal first registers their long-term key, as well as medium-term keys and some cached one-time keys with a key distribution server. Two parties communicate using Signal in long-lived exchanges called *sessions*. A session begins when Alice requests Bob's long-term, medium-term and one-time credentials from a key distribution server (perhaps over an authenticated channel), optionally verifies them out-of-band, and uses them in a proprietary key exchange protocol sometimes called the Signal Key Exchange, "TripleDH" or "X3DH".

The key exchange outputs a master secret, which in turn is used to derive two symmetric keys: a "root key" and a "sending chain key". As messages are sent and received these keys are frequently updated by passing them through a key derivation function (KDF), at the same time deriving output keys which are used elsewhere in the protocol.

When Alice wishes to encrypt a message for Bob, she advances her sending chain by one step, deriving a replacement sending chain key as well as a message encryption key. She can derive subsequent message encryption keys by repeating this process, advancing the sending chain once per message in order to derive a new key. Similarly, when she receives a message from Bob she advances her receiving chain in order to generate a decryption key.

The root chain is advanced through a separate mechanism: when the session is initialised, Alice also generates an ephemeral DH key known as her "ratchet key". She attaches this to her messages, authenticated but not encrypted. When Bob replies to a message, he will send his own "ratchet public key". Upon receiving a new ratchet public key from Bob, Alice advances the root chain *twice*: first with the DH shared secret obtained using her old public key, and second with that using her new. The resulting two outputs of the chain initialise the new receiving and sending chains respectively, and the resulting root chain key replaces the original root chain key.

For the initiator (resp. responder), $mk^{\mathbf{sym\text{-}ir}:x,y}$ denotes the y^{th} symmetric key on the x^{th} sending (resp. receiving) chain, and $mk^{\mathbf{sym\text{-}ri}:x,y}$ the y^{th} symmetric key on the x^{th} receiving (resp. sending) chain. We use the notation **ir** and **ri** for sending and

2. The tagged releases of libsignal lag behind the current codebase. The commit hash of the state of the repository as of our reading is listed in the bibliography. Note that there are separate implementations in C, JavaScript and Java; the latter is used by Android mobile apps and is the one we have read most carefully.

receiving keys from the initiator of the session's perspective: **ir** is from initiator of the session to responder, so corresponds to a sending key for the initiator and receiving key for the responder, and vice versa for **ri**. The notation inherits its complexity from the underlying protocol, but it does allow us to distinctly name each session key that is generated, and will allow us to make note of the subtly different properties of different keys. Thus, we can separate the Signal protocol into four phases:

Registration. (Section 2.3)
: At installation (and periodically afterwards), Alice registers her identity with a key distribution server and uploads some cryptographic data.

Session setup. (Section 2.4)
: Alice requests and receives cryptographic data from Bob (either from the central server or directly from Bob himself), and uses it to setup a long-lived messaging session and establish initial symmetric encryption keys.

Symmetric-ratchet communication. (Section 2.5)
: Alice uses the current symmetric encryption keys of her messaging session for communication with Bob, passing them through a key derivation function on every iteration. The message keys form a type of PRF chain: a "symmetric ratchet".

Asymmetric-ratchet updates. (Section 2.6)
: Alice exchanges DH values with Bob, generating new shared secrets and uses them to begin new chains of message keys. The exchanged DH values give rise to a sequence of shared secrets, which are input with the current key in the "root chain" to the key derivation function to form the "asymmetric ratchet".

Alice and Bob can run many simultaneous sessions between them, each admitting an arbitrary sequence of stages consisting of symmetric and asymmetric ratcheting. We first explain notation and primitives below, and then discuss each of the four phases in detail in subsequent sections.

Table 1 is a glossary to the ten different classes of keys used in the Signal protocol. Figure 1 depicts the operations executed in all stages of the protocol in a pseudocode format.

2.2. Notation and Primitives

Groups. Let g denote the generator of a group G of prime order q; we write the group multiplicatively.

Sessions. We denote A's ith session by π_A^i.

Stages. Within a session, Signal admits a tree of various different stages. We adopt a unified notation to refer to any of them. All stages are described using a term in [square brackets]; the initial stage is always $[0]$ and contains the key exchange. Subsequent stages occur locally at Alice and Bob, but correspond in the sense of generating matching keys.

Alice and Bob assign different roles to the stages they complete: Alice may consider some stage s as generating a sending key, while Bob considers his version of the same stage as generating a receiving key. To avoid persistent case distinctions, we adopt a *role-agnostic* naming scheme, describing stages as "**-ir**" if they are used for the initiator to send to the responder, and as "**-ri**" if they are used for the responder to send to the initiator. This maintains the invariant that stages with the same name generate the same key(s).

asymmetric	ipk_A	ik_A	A's long-term identity key pair
	$prepk_B$	$prek_B$	B's medium-term (signed) prekey pair
	$eprepk_B$	$eprek_B$	B's ephemeral prekey pair
	epk_A	ek_A	A's ephemeral key pair
	$rchpk_A^a$	$rchk_A^a$	A's a^{th} ratchet key pair
symmetric		$ck_A^{\mathbf{sym\text{-}ir}:a,y}$	y^{th} key in A's a^{th} send chain
		$ck_A^{\mathbf{sym\text{-}ri}:a,y}$	y^{th} key in A's a^{th} receive chain
		$mk_A^{\mathbf{sym\text{-}ir}:a,y}$	y^{th} message key in A's a^{th} send chain
		$mk_A^{\mathbf{sym\text{-}ri}:a,y}$	y^{th} message key in A's a^{th} receive chain
		rk_A^a	A's a^{th} root key

TABLE 1: Keys used in the Signal protocol. Asymmetric key pairs show public and private components.

There are two types of asymmetric updates; the first uses a received ratchet key to begin a receiving chain, and the second generates a new ratchet key to begin a sending chain. At a given party, we count the number of asymmetric updates in a variable x; thus, we can refer to the x^{th} update of the first type in a session as stage $[\mathbf{asym\text{-}ri}{:}x]$, and of the second type as $[\mathbf{asym\text{-}ir}{:}x]$. Note that the x^{th} "**-ri**" stage precedes the x^{th} "**-ir**" stage, because the first asymmetric stage is of type "**-ri**".

Similarly, there are two types of symmetric updates, "**-ri**" and "**-ir**", depending on whether the chain to which they belong was created by a stage of type $[\mathbf{asym\text{-}ri}{:}x]$ or $[\mathbf{asym\text{-}ir}{:}x]$. At a given party, we count the number of symmetric updates in the x^{th} symmetric chain in a variable y; thus, we can refer to the y^{th} update in the x^{th} symmetric chain as stage $[\mathbf{sym\text{-}ri}{:}x{,}y]$ or $[\mathbf{sym\text{-}ir}{:}x{,}y]$.

In Signal, for a fixed x, all symmetric stages in which a party generates sending keys in chain x occur before the asymmetric stage $x+1$, but symmetric receiving ratchets in chain x can occur at any time after the parent node in the graph has been established. This accommodates out-of-order message delivery. For example, the initiator performs all stages $[\mathbf{sym\text{-}ir}{:}x{,}1], [\mathbf{sym\text{-}ir}{:}x{,}2], \ldots$ before stage $[\mathbf{asym\text{-}ir}{:}x+1]$, but may delay stages $[\mathbf{sym\text{-}ri}{:}x{,}y]$ as much as necessary.

Keys. Signal distinguishes between at least ten different classes of key, depicted in Table 1, so again for ease of reading we adopt a standardised notation. Keys are written in italics and end with the letter k. For asymmetric key pairs, the corresponding public key ends with the letters *pk*, and is always computed by group exponentiation with base g and the private key as the exponent: $pk = g^k$. If the identity of the agent A who generates a key is unclear we mark this in subscript (i.e. k_A), but omit this where it is clear.

Every stage derives new keys. To identify these keys uniquely, we write the index of the stage deriving a key k in superscript; thus, rk_A^0 would be the root key derived by A in stage $[0]$, and $mk^{\mathbf{sym\text{-}ri}:x,y}$ the message key derived in stage $[\mathbf{sym\text{-}ri}:x,y]$. Not all stages derive all keys: for example, there is no $rk^{\mathbf{sym\text{-}ri}:x,y}$, since root keys are not affected by symmetric updates.

The naming scheme for keys is also role-agnostic: in intended operation, keys will be equal *iff* they have the same name. As with stages, agents have different intended uses for the same key: for example, the initiator would use the key $mk^{\mathbf{sym\text{-}ir}:x,y}$ for encrypting messages to send, and the responder would use the same key for decrypting received messages.

In our model, there are technically no stages $[\mathbf{sym\text{-}ir}:x,0]$ or $[\mathbf{sym\text{-}ri}:x,0]$, but there are keys with these indexes, since the first entry in each sending and receiving chain is created by the asymmetric update starting that chain . We could equivalently think of Signal only deriving message keys in symmetric stages and allowing $y = 0$, in which case asymmetric stages would not derive message keys. Our formulation simply renumbers keys, so that every stage derives a message key.

Cryptographic functions. Signal uses one of two elliptic curves to implement X3DH: curve X25519 or curve X448. The key derivation functions use either $\mathrm{HMAC\text{-}SHA256}$ or HKDF [30] using $\mathrm{SHA256}$.

AEAD denotes an authenticated encryption scheme with associated data. In Signal, this is an encrypt-then-MAC scheme: encryption is $\mathrm{AES256}$ in CBC mode with PKCS#5 padding, and the MAC is $\mathrm{HMAC\text{-}SHA256}$. This is the same combination originally used in TextSecure v3, which was shown by Frosch et al. [22] to have standard authenticated encryption security properties. Since our focus is on the key exchange portion, we omit details of the AEAD and treat it in a black-box fashion.

Sign is related to the Ed25519 signature scheme [5, 43]. Again, we treat it as a black-box signature.

2.3. Registration Phase—Figure 1(a)

Upon installation (and periodically afterwards), all agents generate a number of cryptographic keys and register themselves with a key distribution server.

Specifically, each agent generates the following DH private keys:

(i) a long-term "identity" key *ik*
(ii) a medium-term "signed prekey" *prek*
(iii) multiple short-term "one-time prekeys" *eprek*

The public keys corresponding to these values are then uploaded to the server, together with a signature on *prek* using *ik*.

2.4. Session Setup Phase—Figure 1(b)

In the session-setup phase, public keys are exchanged and used to initialise shared secrets in the session memory. The underlying key exchange protocol is a one-round DH protocol called the Signal Key Exchange or X3DH[3], comprising an exchange of various DH public keys, computation of various DH shared secrets as in Figure 2, and then application of a key derivation function. While many possible variants of such protocols have been explored in-depth in the literature (HMQV [31], Kudla-Paterson [32], NAXOS [33] among many others), the session key derivation used here is new and not based on one of these standard protocols, though it draws some inspiration from [32].

Recall that for asynchronicity Signal uses prekeys: initial protocol messages which are stored at an intermediate server, allowing agents to establish a session with offline peers by retrieving one of their cached messages (in the form of a DH ephemeral public key).

In addition to this ephemeral public key, agents also publish a "medium-term" key, which is shared between multiple peers. This means that even if the one-time ephemeral keys stored at the server are exhausted, the session will go ahead using only a medium-term key. This form of key reuse is studied in [39] and will be modelled in this paper. Thus, session setup in the Signal protocol consists of two steps: first, Alice obtains ephemeral values from Bob (usually via a key distribution server); second, Alice treats the received values as the first message of a Signal key exchange, and completes the exchange in order to derive a master secret.

2.4.1. Receiving ephemerals. The most common way for Alice to receive Bob's session-specific data is for her to query a semi-trusted server for pre-computed values (known as a PreKeyBundle).

When Alice requests Bob's identity information, she receives his identity public key ipk_B, his current signed prekey $prepk_B$, and a one-time prekey $eprepk_B$ if there are any available. Signed pre-keys are stored for the medium term, and therefore shared between everyone sending messages to Bob; one-time keys are deleted by the server upon transmission. Alice's initial message contains identifiers for the prekeys so that Bob can learn which were used.

2.4.2. Building a session. Once Alice has received the above values, she generates her own ephemeral key

3. The key exchange protocol was sometimes referred to as TripleDH, from the three DH shared secrets always used in the KDF (although in most configurations four shared secrets are used). The name QuadrupleDH has also been used for the variant which includes the long-term/long-term DH value, not as might be expected the variant which includes the one-time prekey.

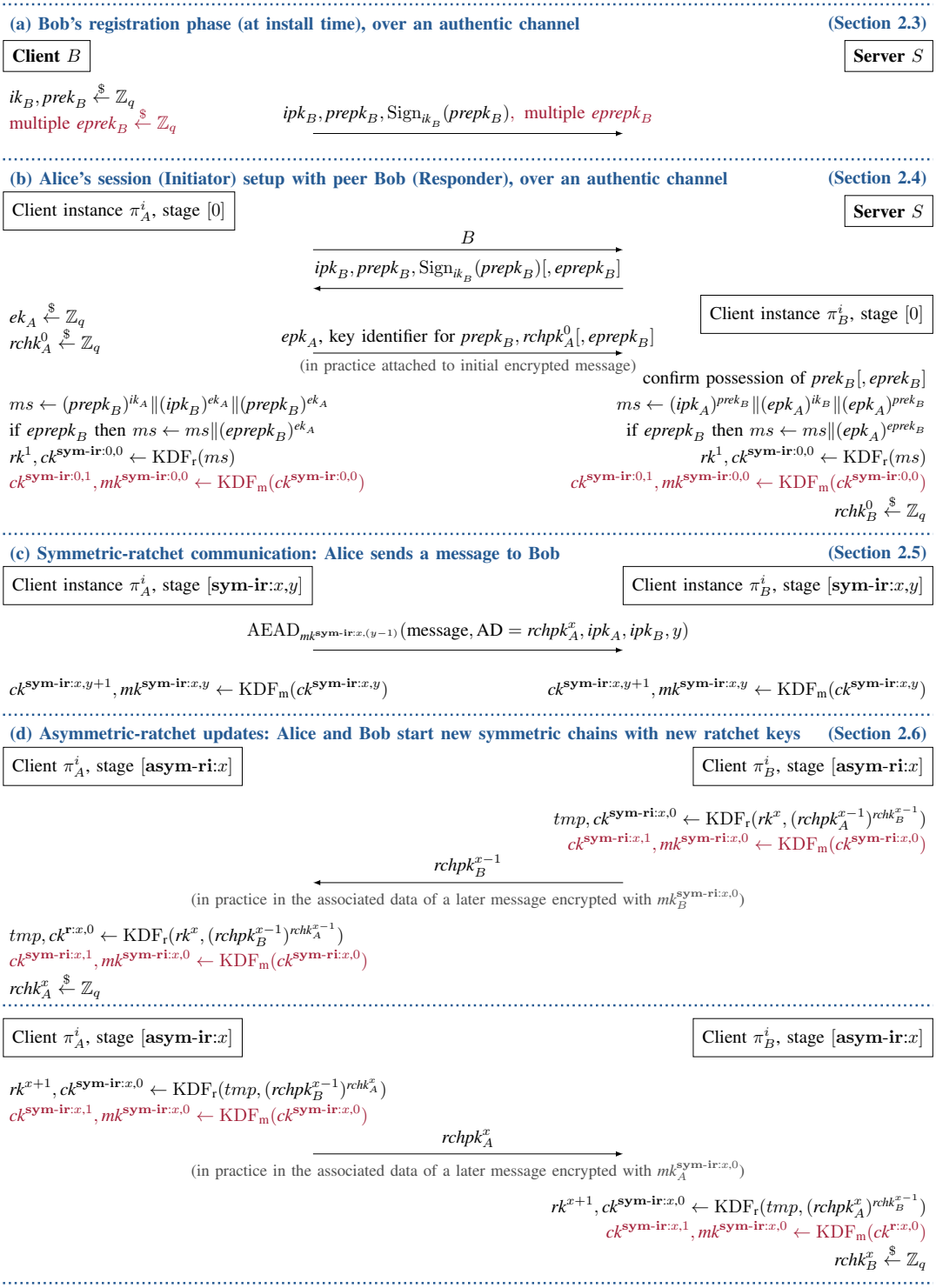

Figure 1: Signal protocol including preregistration of keys. Local actions are depicted in the left and right columns, and messages flow between them. We show only one step of the symmetric and asymmetric ratchets; they can be iterated arbitrarily. Variables storing keys are defined in Table 1, and session identifiers in Table 2. Dark red text indicates reordered actions in our model, as discussed in Section 5. Each stage derives message keys with the same index as the stage number, and chaining/root keys with the index for the next stage; the latter is passed as state from one stage to the next. State info st in asymmetric stages is defined as the root key used in the key derivation, and for symmetric stages st is defined as the chain key used in key derivation. Symmetric stages always start at $y = 1$ and increment. When an actor sends consecutive messages, the first message is a DH ratchet and then subsequent messages use the symmetric ratchet. When an actor replies, they always DH ratchet first; they never carry on the symmetric ratchet.

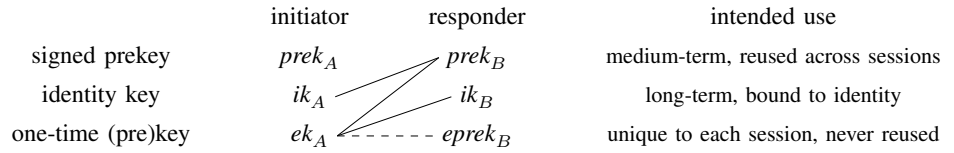

Figure 2: Diffie–Hellman private keys used in the Signal Key Exchange KDF. An edge between two private keys (e.g., ik_A and $prek_B$) indicates that their DH value ($g^{ik_A \cdot prek_B}$) is included in the final KDF computation. The dashed line is optional: it is omitted from the session key derivation if $eprek_B$ is not sent. Note the asymmetry: when Alice initiates a session with Bob, her signed prekey is not used at all. Our freshness conditions in Section 4.3 on page 10 will be partially based on this graph.

ek_A, and computes a session key by performing three or four group exponentiations as depicted in Figure 2. She then concatenates the resulting shared secrets and passes them through a key derivation function to derive an initial root key rk^0 and sending chain key $ck^{\mathbf{s}:0,0}$. (No DH value is passed to $\mathrm{KDF_r}$ for this initial invocation.) For modelling purposes, we also have Alice generate her initial sending message key $mk^{\mathbf{sym\text{-}ir}:0,0}$ (which is this stage's session key output) and the next sending chain key $ck^{\mathbf{sym\text{-}ri}:0,0}$. Finally, she generates a new ephemeral DH key $rchk^0$ known as her ratchet key.

For Bob to complete[4] the key exchange, he must receive Alice's public ephemeral key epk_A. In the Signal protocol, Alice attaches this value to all messages that she sends (until she receives a message from Bob, since from such a message she can conclude that Bob received epk_A). To disentangle the stages of the model, we have Alice send epk_A in a separate message; thus, once the session-construction stage is complete, both Alice and Bob have derived their root and chain keys.

When Bob receives epk_A, he first checks that he currently knows the private keys corresponding to the identity, signed pre-, and one-time pre-key which Alice used. If so, he performs the receiver algorithm for the key exchange, deriving the same root key rk^0 and chain key (which he records as $ck^{\mathbf{r}:0,0}$). For modelling purposes, we also have Bob generate his initial receiving message key $mk^{\mathbf{sym\text{-}ir}:0,0}$ (which is this stage's session key output) and the next receiving chain key $ck^{\mathbf{sym\text{-}ir}:0,0}$.

2.5. Symmetric-Ratchet Phase—Figure 1(c)

Two sequences of symmetric keys will be derived using a PRF chain, one for sending and one for receiving. The symmetric chainsmay be advanced for one of two reasons: either Alice wishes to send a new message, or she wishes to decrypt a message she has just received.

In the former case, Alice takes her current sending chain key $ck^{\mathbf{sym\text{-}ir}:x,y}$ and applies the message key derivation function $\mathrm{KDF_m}$ to derive two new keys: an updated sending chain key $ck^{\mathbf{sym\text{-}ir}:x,(y+1)}$ and a sending message key $mk^{\mathbf{sym\text{-}ir}:x,y}$. Alice uses the sending message key to encrypt her outgoing message, then deletes it and the old sending chain key. This process can be repeated arbitrarily.

When Alice receives an encrypted message, she checks the accompanying ratchet public key to confirm that she has not yet processed it, and if not she then performs an asymmetric ratchet update, described below. Regardless, she then reads the metadata in the message header to determine the index of the message in the receiving chain, and advances the receiving chain as many steps as is necessary to derive the required receiving message key; by construction, Alice's receiving message keys equal Bob's sending keys. Unlike for the sending case, Alice cannot delete receiving message keys immediately; she must wait to receive a message encrypted under each one. (Otherwise, out-of-order messages would be undecryptable since their keys would have been deleted.) The open source implementation of Signal has a hard-coded limit of 2000 messages or five asymmetric updates, after which old keys are deleted even if they have not yet been used.

2.6. Asymmetric-Ratchet Phase—Figure 1(d)

The final top-level phase of Signal is the asymmetric-ratchet update. In this phase, Alice and Bob take turns generating and sending new DH public keys and using them to derive new shared secrets. These are accumulated in the asymmetric ratchet chain, from which new (symmetric) message chains are initialized.

When Alice receives a message from Bob, it may be accompanied by a new (previously unseen) ratchet public key $rchpk_B^{x-1}$. If so, this triggers Alice to enter her next asymmetric ratchet phase $[\mathbf{asym\text{-}ir}:x]$. Note that Alice already has stored a previously generated private ratchet key $rchk_A^{x-1}$. Before decrypting the message, Alice updates her asymmetric ratchet as per Figure 1. This consists of two steps. In the first step, denoted $rchk_A^x$, deriving two DH shared secrets $[\mathbf{asym\text{-}ri}:x]$, she computes a first DH shared secret (between the received ratchet public key and her old ratchet private key), and combines this with the root chain key to derive a new receiving chain key and receiving message key. In the second step, denoted $[\mathbf{asym\text{-}ir}:x]$, she computes a second DH shared secret

4. If the initial message from Alice is invalid, Bob will in fact not complete a session. This does not affect our analysis, which considers only secrecy of session keys, but may become important if e.g. analysing deniability.

(between the received ratchet public key and her new ratchet private key), and combines this with the root chain key and the first DH shared secret to derive a new sending chain key and sending message key, as well as the root chain key for the next asymmetric stage.

The message keys in the first and second steps have slightly different security properties, so they are recorded in our model as belonging to distinct stages $[\mathbf{asym\text{-}ri}{:}x]$ and $[\mathbf{asym\text{-}ir}{:}x]$.

Alice then sends her new ratchet public key $rchpk_A^x$ along with future messages to Bob, and the process continues indefinitely.

Bob does the corresponding operations shown in Figure 1 to compute the same DH shared secrets and the corresponding root, chain, and message keys. While symmetric updates can be triggered either by Alice (the session initiator) or Bob (the session responder) and thus could be as in Figure 1(c) or its horizontal flip, asymmetric updates can only be triggered by Alice (the session initiator) receiving a new (previously unseen) ratchet key from Bob (the session responder) and not the other way around, so Figure 1(d) will never be horizontally flipped.

3. Threat Models

We will analyze Signal in the context of a fully adversarially-controlled network. The high-level properties we aim to prove are secrecy and authentication of message keys. Authentication will be implicit (only the intended party could compute the key) rather than explicit (proof that the intended party did compute the key). Forward secrecy and "future" secrecy are not explicit goals; instead, derived session keys should remain secret under a variety of compromise scenarios, including if a long-term secret has been compromised but a medium or ephemeral secret has not (forward secrecy) or if state is compromised and then an uncompromised asymmetric stage later occurs ("future" or post-compromise secrecy [11]). We assume out-of-band verification of identity keys and medium-term keys, and do not consider side channel attacks.

The finer details of our threat model are ultimately encoded in the so-called freshness predicate, specified in Section 4.3 on page 10, where we provide further information on our threat model design choices.

On Our Choice of Threat Model. Because at the time of writing Signal did not claim any formally-specified security properties, as part of our analysis we had to decide which threat model to assume. The README document accompanying the source code [36] states that Signal "is a ratcheting forward secrecy protocol that works in synchronous and asynchronous messaging environments". A separate GitHub wiki page [42] provides some more goals (forward and future secrecy[5], metadata encryption and detection of message replay/reorder/deletion) but to the best of our knowledge no mention of message integrity or authentication is made other than the use of AEAD cipher modes.

We believe that the threat model we have chosen is realistic, although we discuss later some directions in which it could be strengthened. As is common in the AKE literature, we assume a trusted public key infrastructure (PKI). Since in Signal the PKI is combined with the network (the same server distributes identity and ephemeral keys), this assumption somewhat restricts the attacker's control over particular sessions.

Some claims have been made about privacy and deniability [49] in Signal, but these are relatively abstract. In general, signatures are used but only for the signed pre-key in the initial handshake, meaning that an observer can prove that Alice sent a message [16, full deniability] to *someone* but perhaps not to *Bob* [13, peer deniability].

Additionally, one might consider a threat model that includes imperfect ephemeral random number generators. Since no static-static DH shared secret is included in Signal's KDF, an adversary who knows all ephemeral values can compute all secrets. However, Signal continuously updates state with random numbers, so we capture in our threat model the fact that it is possible to make some security guarantees, if some, but not all, random numbers are compromised.

The trust assumptions on the registration channel are not defined; Signal specifies a mandatory method for participants to verify each other's identity keys through an out-of-band channel, but most implementations do not require such verification to take place before messaging can occur. Without it, an untrusted key distribution server can impersonate any agent.

Signal's mechanisms suggest a lot of effort has been invested to protect against the loss of secrects used in specific communications. If the corresponding threat model is an attacker gaining (temporary) access to the device, it becomes crucial if certain previous secrets and decrypted messages can be accessed by the attacker or not: generating new message keys is of no use if the old ones are still recoverable. This, in turn, depends on whether *deletion* of messages and previous secrets has been effective. This is known to be a hard problem, especially on flash-based storage media [46], which are commonly used on mobile phones.

4. Security Model

In this section we present a security model for multi-stage key exchange, which we then apply to model Signal's initial key exchange as well as its ratcheting scheme. Our model allows multiple parties to execute multiple, concurrent *sessions*; each session has multiple *stages*. For Signal, the session represents a single chat between two parties, and each stage represents a new application of the ratchet. Figure 1 depicts, roughly, a single session. There are three types of stage in Signal: the initial key exchange, asymmetric ratcheting, and symmetric ratcheting. In addition, ratcheting stages differ based on whether they

5. Future secrecy means "a leak of keys to a passive eavesdropper will be healed by introducing new DH ratchet keys" [42].

are used for generating keys for the initiator to send to the responder (denoted -ir) or vice versa (denoted -ri). For our purposes, every stage generates a session key; depending on the stage, this will be either the sending or the receiving message key.

On the choice of model. We choose to study the security of Signal in the traditional key exchange notion of *key indistinguishability* (albeit a multi-stage variant), as opposed to a monolithic *secure channel* notion such as ACCE [25]. This is possible because Signal does not use session keys in the channel establishment.

Model notation. We present our model as a pseudocode experiment where the primitive in question (the multi-stage key exchange protocol) is modelled as a tuple of algorithms, and then an adversary interacts with the experiment. This approach is commonly used in many other areas of cryptography, but less so in key exchange papers. Compared with models and experiments presented in textual format, we argue that our approach makes it easier to understand some precise details, and easier to see subtleties in variations.

We adopt the following notational and typographic conventions. `Monotype` text denotes constants; serif text denotes algorithms and variables associated with the actual protocol (variables are *italicized*); and sans-serif text denotes algorithms, oracles, and variables associated with the experiment. Algorithms and Oracles start with upper-case letters; variables start with lower-case letters. We use object-oriented notation to represent collections of variables. In particular, we will use π_u^i to denote the collection of variables that party u uses in its i^{th} protocol execution ("session"). To denote the variable v in stage s of party u's ith session, we write $\pi_u^i.v[s]$; note s is not (necessarily) a natural number. For Signal, s is [0] for the session setup stage; [**sym-ir**:x,y] or [**sym-ri**:x,y] for symmetric sending or receiving stages; or [**asym-ri**:x] or [**asym-ir**:x] for the 1st and 2nd portions of the xth asymmetric stage. (See also Figure 1.)

DH protocols conventionally use both ephemeral keys (unique to a session) and long-term keys (in all sessions of an agent). Signal's prekeys do not fit cleanly into this separation, and in order to follow the conventions of the field we refer to the reused DH keys as "medium-term keys".

Medium-term key authentication. Signal's medium-term keys are signed by the same identity key used for DH, breaking key separation. Although there has been some analysis of this form of key reuse [15, 41], it is nontrivial to prove secure. We instead enforce authentication by fiat, allowing the adversary to select any of the medium-term keys owned by an agent, but not to inject their own. In the game, this is implemented as an extra argument when the adversary creates a new session.

4.1. Multi-Stage Key Exchange Protocol

Definition 1 (Multi-stage key exchange protocol)**.** A *multi-stage key exchange protocol* Π is a tuple of algorithms, along with a keyspace $\mathcal{K}$ and a security parameter λ indicating the number of bits of randomness each session requires. The algorithms are:

- $\text{KeyGen}() \xmapsto{\$} (\textit{ipk}, \textit{ik})$: A probabilistic *long-term key generation algorithm* that outputs a long-term public key / secret key pair $(\textit{ipk}, \textit{ik})$. In Signal, these are called "identity keys".
- $\text{MedTermKeyGen}(\textit{ik}) \xmapsto{\$} (\textit{prepk}, \textit{prek})$: A probabilistic *medium-term key generation algorithm* that takes as input a long-term secret key *ik* and outputs a medium-term public key / secret key pair $(\textit{prepk}, \textit{prek})$. In Signal, these are called "signed prekeys"; in the key exchange literature, they are sometimes called "semi-static keys".
- $\text{Activate}(\textit{ik}, \textit{prek}, \textit{role}) \rightarrow (\pi', m')$: A probabilistic *protocol activation algorithm* that takes as input a long-term secret key *ik*, a medium-term secret key *prek*, and a role $\textit{role} \in \{\texttt{init}, \texttt{resp}\}$, and outputs a state π' and (possibly empty) outgoing message m'.
- $\text{Run}(\textit{ik}, \textit{prek}, \pi, m) \rightarrow (\pi', m')$: A probabilistic *protocol execution algorithm* that takes as input a long-term secret key *ik*, a medium-term secret key *prek*, a state π, and an incoming message m, and outputs an updated state π' and (possibly empty) outgoing message m'.

Definition 2 (State)**.** A *state* π is a collection of the following variables:

- $\pi.\textit{role} \in \{\texttt{init}, \texttt{resp}\}$: the instance's role
- $\pi.\textit{peeripk}$: the peer's long-term public key
- $\pi.\textit{peerprepk}$: the peer's medium-term public key
- $\pi.\textit{status}[s] \in \{\varepsilon, \texttt{active}, \texttt{accept}, \texttt{reject}\}$: execution status for stage s, set to `active` upon start of a new stage, and set to `accept` or `reject` by computation of the stage's ratchet key.
- $\pi.k[s] \in \mathcal{K}$: the session key output by stage s
- $\pi.\textit{st}[s]$: any additional protocol state values that a previous stage gives as input to stage s (defined as part of the protocol).
- $\pi.\textit{sid}[s]$: the identifier of stage s of session π; this is view the actor has of the session π in stage s, as defined in Figure 1.
- $\pi.\textit{type}[s]$: the type of freshness required for this stage to have security. For Signal, this is `triple`, `triple+DHE`, `asym-ir`, `asym-ri`, `sym-ir` or `sym-ri`.

The state of an instance π in our experiment models "real" protocol state that an implementation would keep track of and use during protocol execution. We will supplement this in the experiment with additional variables that are artificially added for the experiment. These are administrative identifiers, used to formally reason about what is happening in our security experiment, e.g., to identity sessions and partners.

4.2. Key Indistinguishability Experiment

Having defined what a multi-stage key exchange protocol is, we can now define the experiment for key indistinguishability.

As is typical in key exchange security models, the experiment establishes long-term keys and then allows the adversary to interact with the system. The adversary can direct parties to start sessions with particular medium-term keys, and can control the delivery of messages to parties (including modifying, dropping, delaying, and inserting messages). The adversary can learn various long-term or per-session secret information from parties via reveal queries, and at any point can choose a single stage of a single session to "test". They are then given either the real session key from this stage, or a random key from the same keyspace, and asked to decide which was given. If they decide correctly, we say they win the experiment. This is formalized in the following definition and corresponding experiment.

Definition 3 (Multi-stage key indistinguishability)**.** Let Π be a key exchange protocol. Let $\mathsf{n_P}, \mathsf{n_M}, \mathsf{n_S}, \mathsf{n_s} \in \mathbb{N}$. Let $\mathcal{A}$ be a probabilistic algorithm that runs in time polynomial in the security parameter. Define

$$\mathsf{Adv}^{\mathsf{ms\text{-}ind}}_{\Pi,\mathsf{n_P},\mathsf{n_M},\mathsf{n_S},\mathsf{n_s}}(\mathcal{A}) = \Pr\left[\mathsf{Exp}^{\mathsf{ms\text{-}ind}}_{\Pi,\mathsf{n_P},\mathsf{n_M},\mathsf{n_S},\mathsf{n_s}}(\mathcal{A}) = 1\right] - 1/2$$

where the security experiment $\mathsf{Exp}^{\mathsf{ms\text{-}ind}}_{\Pi,\mathsf{n_P},\mathsf{n_M},\mathsf{n_S},\mathsf{n_s}}(\mathcal{A})$ is as defined in Figure 3. Note $\mathsf{n_S}$ and $\mathsf{n_s}$ are upper bounds on the number of sessions per party and number of stages per session that can be established. We call an adversary efficient if it runs in time polynomial in the security parameter.

We are working in the post-specified peer model, where the peer's identity is learned by the actor during the execution of the protocol, by virtue of learning the peer's public key; and similarly for the peer's semi-static key. Certain aspects of the experiment require the administrative index of the corresponding key, and thus, we assume that $\pi^i_u.\mathsf{peerid}$ is set to the corresponding index upon $\pi^i_u.\mathit{peeripk}$ being set; and similarly for the semi-static key index $\pi^i_u.\mathsf{peerpreid}$ upon $\pi^i_u.\mathit{peerprepk}$ being set. (Recall that experiment-only variables are in sans-serif.)

4.2.1. Session identifiers. We define the session identifiers $sid[s]$ for each stage $[s]$ of Signal in Table 2. It is important to note that these session identifiers only exist in our model, not in the protocol specification itself. We use them to we define restrictions on the adversary's allowed behaviour in our model, so that we can make precise security statements: we will generally restrict the adversary from making queries against any session with the same session identifier as the Test session. If two sessions have equal session identifiers we say that they are "matching".

The precise components of the session identifiers are crucial to our definition of security: the more information is included in the session identifier, the more specific the restriction on the adversary and hence the stronger the security model. In particular, we do *not* include identities, because they are not included in Signal's key derivation or associated data of encrypted messages. This means that the unknown key share attack against TextSecure [22] is not considered an attack in our model: Alice's session with Eve will have the same session identifier as Bob's session with Alice.

4.3. Freshness

From a key exchange perspective, the novelty of Signal is the different security goals of different stages' session keys. The subtle differences between those security goals are captured in the details of the threat model. Previously, we provided the adversary with powerful queries with which it can break any protocol. We now define the so-called freshness predicate fresh to constrain that power, effectively specifying the details of the threat model.

Our goal of the fresh predicate is to describe the best security condition that might be provable for each of Signal's message keys based on the protocol's design; here, "best" is with respect to the maximal combinations of secrets learned by the adversary.

The main motivation for our fresh predicate for the initial stages comes from observing Figure 2 on page 7. In the graph, the edges can be seen as the individual secrets established between initiator and responder, on which the secrecy of the session keys is based. If the adversary cannot learn the secret corresponding to one of these edges, it cannot compute the session key. The adversary can learn the secret corresponding to an edge if it can compromise one of the two endpoints. Thus, if an adversary can learn, e.g., the initiator A's ik_A and ek_A, it can derive the secrets corresponding to all edges. A similar observation can be made for the responder. However, after keys are updated, the situation changes, since additional secrets are introduced, and the adversary may no longer have enough information. We define modified freshness conditions for subsequent stages to capture Signal's post-compromise security properties.

We define our freshness conditions to exactly exclude those cases for which we can directly observe that the protocol design offers no protection. For example, the design does not include an edge between the long-term keys of the parties (sometimes referred to as the *static Diffie-Hellman key*). This implies that if the long-term keys are secret, but all generated randomness is compromised, Signal offers no protection, since all edges become compromised. Because our freshness conditions are based on fig. 2 and the subsequent key updates, we do not consider this scenario an attack but rather a direct consequence of the design. In this work we aim to prove that, working from the design choices made, Signal indeed achieves the best it can (without introducing further elements in the key derivation function).

The freshness predicate fresh for our experiment works hand-in-hand with a variety of sub-predicates ($\mathsf{clean}_{\mathtt{triple}}$, $\mathsf{clean}_{\mathtt{triple+DHE}}$, $\mathsf{clean}_{\mathtt{asym\text{-}ir}}$, $\mathsf{clean}_{\mathtt{asym\text{-}ri}}$, $\mathsf{clean}_{\mathtt{sym\text{-}ir}}$ and $\mathsf{clean}_{\mathtt{sym\text{-}ri}}$) which are highly specialized to Signal to capture the exact type of security achieved Signal's different types of stages. We define cleanness below.

$\underline{\mathsf{Exp}^{\mathsf{ms\text{-}ind}}_{\Pi,\mathsf{n_P},\mathsf{n_M},\mathsf{n_S},\mathsf{n_s}}(\mathcal{A})}$:

1: $\mathsf{b} \xleftarrow{\$} \{0,1\}$
2: $\mathsf{tested} \leftarrow \bot$
3: *// generate long-term and semi-static keys*
4: **for** $u = 1$ **to** $\mathsf{n_P}$ **do**
5: $\quad (ipk_u, ik_u) \xleftarrow{\$} \mathrm{KeyGen}()$
6: $\quad$ **for** $\mathsf{preid} = 1$ **to** $\mathsf{n_M}$ **do**
7: $\quad\quad (prepk_u^{\mathsf{preid}}, prek_u^{\mathsf{preid}}) \xleftarrow{\$} \mathrm{MedTermKeyGen}(ik_u)$
8: $pubinfo \leftarrow (ipk_1, \ldots, ipk_{\mathsf{n_P}}, prepk_1^1, \ldots, prepk_{\mathsf{n_P}}^{\mathsf{n_M}})$
9: $\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathsf{Send},\mathsf{Rev}*,\mathsf{Test}}(pubinfo)$
10: **if** $(\mathsf{tested} \neq \bot) \wedge \mathsf{fresh}(\mathsf{tested}) \wedge \mathsf{b} = \mathsf{b}'$ **then**
11: $\quad$ **return** 1 *// the adversary wins*
12: **else**
13: $\quad$ **return** 0 *// the adversary loses*

$\underline{\mathsf{Send}(u,i,m)}$:

1: **if** $\pi_u^i = \bot$ **then**
2: $\quad$ *// start new session and record intended peer*
3: $\quad$ parse m as $(\pi_u^i.\mathsf{preid}, role)$
4: $\quad \pi_u^i.\vec{\mathsf{rand}} \xleftarrow{\$} \{0,1\}^{\mathsf{n_S} \times \lambda}$
5: $\quad (\pi_u^i, m') \leftarrow \mathrm{Activate}(ik_u, prek_u^{\pi_u^i.\mathsf{preid}}, role; \pi_u^i.\mathsf{rand}[0])$
6: $\quad$ **return** m'
7: **else**
8: $\quad s \leftarrow \pi_u^i.stage$
9: $\quad (\pi_u^i, m') \leftarrow \mathrm{Run}(ik_u, prek_u^{\pi_u^i.\mathsf{preid}}, \pi_u^i, m; \pi_u^i.\mathsf{rand}[s])$
10: $\quad$ **return** m'

$\underline{\mathsf{RevSessKey}(u,i,s)}$:

1: $\pi_u^i.\mathsf{rev\_session}[s] \leftarrow \mathrm{true}$
2: **return** $\pi_u^i.k[s]$

$\underline{\mathsf{RevLongTermKey}(u)}$:

1: $\mathsf{rev\_ltk}_u \leftarrow \mathrm{true}$
2: **return** ik_u

$\underline{\mathsf{RevMedTermKey}(u,\mathsf{preid})}$:

1: $\mathsf{rev\_mtk}_u^{\mathsf{preid}} \leftarrow \mathrm{true}$
2: **return** $prek_u^{\mathsf{preid}}$

$\underline{\mathsf{RevRand}(u,i,s)}$:

1: $\pi_u^i.\mathsf{rev\_random}[s] \leftarrow \mathrm{true}$
2: **return** $\pi_u^i.\mathsf{rand}[s]$

$\underline{\mathsf{RevState}(u,i,s)}$:

1: $\pi_u^i.\mathsf{rev\_state}[s] \leftarrow \mathrm{true}$
2: **return** $\pi_u^i.st[s]$

$\underline{\mathsf{Test}(u,i,s)}$:

1: *// can only call* Test *once, and only on accepted stages*
2: **if** $(\mathsf{tested} \neq \bot)$ **or** $(\pi_u^i.status[s] \neq \mathtt{accept})$ **then**
3: $\quad$ **return** $\bot$
4: $\mathsf{tested} \leftarrow (u,i,s)$
5: *// return real or random key depending on b*
6: **if** $\mathsf{b} = 0$ **then**
7: $\quad$ **return** $\pi_u^i.k[s]$
8: **else**
9: $\quad k' \xleftarrow{\$} \mathcal{K}$
10: $\quad$ **return** k'

Figure 3: Security experiment for adversary $\mathcal{A}$ against multi-stage key indistinguishability security of protocol Π.

Definition 4 (Freshness)**.** Define the predicate fresh as follows:

$$\begin{aligned}\mathsf{fresh}(u,i,s) = (&\pi_u^i.status[s] = \mathtt{accept})\\ &\wedge \neg \pi_u^i.\mathsf{rev\_session}[s]\\ &\wedge \big(\forall\ j : \pi_u^i.sid[s] = \pi^j_{\pi_u^i.\mathsf{peerid}}.sid[s]\\ &\quad \implies \neg \pi^j_{\pi_u^i.\mathsf{peerid}}.\mathsf{rev\_session}[s]\big)\\ &\wedge \mathsf{clean}_{\pi_u^i.type[s]}(u,i,s)\end{aligned}$$

fresh and its sub-clauses have access to all variables in the experiment (global, user, session, and stage).

4.3.1. Session Setup Stage [0]**.** Intuitively, a triple-DH or triple-DH+DHE key exchange should be secure as long as at least one of its DH shared secrets is secure; thus the clauses of $\mathsf{clean}_{\mathtt{triple}}$ and $\mathsf{clean}_{\mathtt{triple+DHE}}$ correspond to those components. Note that $\mathsf{clean}_{\mathtt{triple}}$ and $\mathsf{clean}_{\mathtt{triple+DHE}}$ only need to be defined for the initial key exchange, i.e., stage [0].

Definition 5 ($\mathsf{clean}_{\mathtt{triple}}$)**.** Within the same context as Definition 4, define

$$\begin{aligned}\mathsf{clean}_{\mathtt{triple}}(u,i,[0]) = &\ \mathsf{clean}_{\mathtt{LM}}(u,i)\\ &\vee \mathsf{clean}_{\mathtt{EL}}(u,i,0)\\ &\vee \mathsf{clean}_{\mathtt{EM}}(u,i,0)\end{aligned}$$

Definition 6 ($\mathsf{clean}_{\mathtt{triple+DHE}}$)**.** Within the same context as Definition 4, define

$$\begin{aligned}\mathsf{clean}_{\mathtt{triple+DHE}}(u,i,[0]) = &\ \mathsf{clean}_{\mathtt{triple}}(u,i,[0])\\ &\vee \mathsf{clean}_{\mathtt{EE}}(u,i,0,0)\end{aligned}$$

For the sub-clauses $\mathsf{clean}_{\mathtt{XY}}$ in the above two definitions, our convention is that initiator's key is of type X and the responder's key of type Y, where the possible types are L, M, and E for long-term (*ik*), medium-term (*prek*), and ephemeral (*ek*) keys respectively, as in Figure 2. This necessitates the two definitions below of $\mathsf{clean}_{\mathtt{LM}}/\mathsf{clean}_{\mathtt{EL}}/\mathsf{clean}_{\mathtt{EM}}$ for when the tested session is the initiator or responder.

These three definitions are straightforward for initiator sessions. For responder sessions, the difficult part is that the ephemeral key is now the peer's, not the actor's: to ensure that it is not known by the adversary, we have to ensure the peer session's randomness has not been revealed (identifying the peer session using session identifiers), and that key actually has to come from an honest peer (meaning the peer session must exist). The following clause helps identify that precise situation:

$$\begin{aligned}&\mathsf{clean}_{\mathtt{peerE}}(u,i,s) =\\ &\quad \Big(\forall\ j : \pi_u^i.sid[s] = \pi^j_{\pi_u^i.\mathsf{peerid}}.sid[s]\\ &\qquad\quad \implies \neg \pi^j_{\pi_u^i.\mathsf{peerid}}.\mathsf{rev\_random}[s]\Big)\\ &\quad \wedge \exists\ j : \pi_u^i.sid[s] = \pi^j_{\pi_u^i.\mathsf{peerid}}.sid[s]\end{aligned}$$

We can now define our various clean predicates. This is a non-trivial restriction on the adversary. If the medium-term key is corrupted then we do not permit an attack impersonating Alice to Bob: since the only randomness in a X3DH handshake is from the initiator (and there is no static-static DH secret), such an attack

name	sid	
$sid[0]$	$(\texttt{triple} : ipk_i, ipk_r, prepk_r, epk_i)$	if $type[0] = \texttt{triple}$
	$(\texttt{triple+DHE} : ipk_i, ipk_r, prepk_r, epk_i, eprepk_r)$	if $type[0] = \texttt{triple+DHE}$
$sid[\mathbf{asym\text{-}ri}{:}x]$	$sid[0] \parallel (\texttt{asym-ri} : rchpk_i^0, rchpk_r^0)$	if $x = 1$
	$sid[\mathbf{asym\text{-}ir}{:}x-1] \parallel (\texttt{asym-ri} : rchpk_r^{x-1})$	if $x > 1$
$sid[\mathbf{asym\text{-}ir}{:}x]$	$sid[\mathbf{asym\text{-}ri}{:}x] \parallel (\texttt{asym-ir} : rchpk_i^x)$	if $x > 0$
$sid[\mathbf{sym\text{-}ri}{:}x,y]$	does not exist	if $x = 0$
	$sid[\mathbf{asym\text{-}ri}{:}x] \parallel (\texttt{sym-ri} : y)$	if $x > 0$
$sid[\mathbf{sym\text{-}ir}{:}x,y]$	$sid[0] \parallel (\texttt{sym} : y)$	if $x = 0$
	$sid[\mathbf{asym\text{-}ir}{:}x] \parallel (\texttt{sym-ir} : y)$	if $x > 0$

TABLE 2: Definition of session identifiers $sid[s]$ for an arbitrary stage s. Since our stages are named role-agnostically, the definitions for initiator and responder stages coincide; we use i to refer to the identity of the initiator and r for that of the responder. For example, if Alice believes she is responding to Bob, then ipk_i denotes Bob's identity public key and ipk_r denotes Alice's. The initial asymmetric stage sid contains two ratchet keys (instead of one) since they are not used in the initial session key derivation and thus are not contained in $sid[0]$. We note that $sid[\mathbf{sym\text{-}ir}{:}x,y]$ for $x = 0$ does not exist because the receiver never starts a symmetric chain immediately after the handshake, always first performing a DH ratchet.

will succeed.

Since we reveal randomness instead of specific keys, this final predicate applies to both the ephemeral keys and the ratchet keys, a fact which we shall use later when defining cleanness of asymmetric stages.

4.3.2. Asymmetric Stages $[\mathbf{asym\text{-}ir}{:}x]/[\mathbf{asym\text{-}ri}{:}x]$. In Signal, keys are updated via either symmetric or asymmetric ratcheting. Asymmetric ratcheting introduces new DH shared secrets into the state, whereas symmetric ratcheting solely applies a KDF to existing state.

It will be helpful to have the following predicate:

Definition 7 ($\mathsf{clean}_{\mathsf{state}}$)**.** Within the same context as Definition 4, define

$$\begin{aligned}\mathsf{clean}_{\mathsf{state}}(u,i,s,s') = \neg\pi_u^i.\mathsf{rev\_state}[s] \\ \wedge \big(\forall\ j : \pi_u^i.sid[s] = \pi^j_{\pi_u^i.\mathsf{peerid}}.sid[s'] \\ \implies \neg\pi^j_{\pi_u^i.\mathsf{peerid}}.\mathsf{rev\_state}[s']\big)\end{aligned}$$

For brevity, we will write $\mathsf{clean}_{\mathsf{state}}(u,i,s)$ as a shorthand for $\mathsf{clean}_{\mathsf{state}}(u,i,s,s)$.

The state reveal query reveals additional state information that a previous stage gives as input to stage s. For Signal, we define as follows. For asymmetric stages, state reveal gives the root key used in the session key computation that was derived in the previous stage; for symmetric stages, state reveal gives the chain key derived in the previous stage.

During asymmetric ratcheting, there are actually two substages, in which keys with slightly different properties are derived. In the first substage, the parties apply a KDF to two pieces of keying material: the root key derived at the end of the previous asymmetric stage, and a DH shared secret derived from both party's previous ratcheting public keys. Keys from this substage are marked with $sid[\mathbf{asym\text{-}ri}{:}x]$; they should be secure if either of the two pieces is unrevealed, which is what type `asym-ri` captures. In the second substage, the parties effectively apply a KDF to three pieces of keying material: the root key, a DH shared secret from the first substage, and a DH shared secret derived from one party's previous ratcheting public key and the other's new ratcheting public key. Keys from this substage are marked with $sid[\mathbf{asym\text{-}ir}{:}x]$ and should be secure if at least one of the three pieces is unrevealed, which is what `asym-ir` captures.
These clauses capture the "future secrecy" goal of Signal: if a device had been compromised at some prior time (i.e., the party's long-term key, past states, and past ephemeral keys are all compromised, and thus neither the second disjuncts nor $\mathsf{clean}_{\mathtt{EE}}(u,i,s'_{ir},s'_{ri})$ are satisfied), but the current ephemeral keys of both parties are uncompromised and honest ($\mathsf{clean}_{\mathtt{EE}}(u,i,s_{ir},s_{ri})$ *is* satisfied) then the stage is clean. Since our security property requires that the adversary cannot learn keys derived from clean stages, this captures post-compromise security.

Note that $\mathsf{clean}_{\mathtt{EE}}$ is used twice (because cleanness of ephemerals is defined as cleanness of the random numbers): once to show that the randomness is clean when generating ephemerals for the initial key exchange, and once to show that it is clean when generating the first ratchet key pair.

4.3.3. Symmetric Stages $[\mathbf{sym\text{-}ir}{:}x,y]$ and $[\mathbf{sym\text{-}ri}{:}x,y]$. For stages with only symmetric ratcheting, new session keys should be secure only if the state is unknown to the adversary: this demands

$$\mathsf{clean}_{\mathtt{LM}}(u,i) = \begin{cases} \neg\mathsf{rev\_ltk}_u \wedge \neg\mathsf{rev\_mtk}^{\pi_u^i.\mathsf{peerpreid}}_{\pi_u^i.\mathsf{peerid}} & \pi_u^i.\mathit{role} = \mathtt{init} \\ \neg\mathsf{rev\_ltk}_{\pi_u^i.\mathsf{peerid}} \wedge \neg\mathsf{rev\_mtk}^{\pi_u^i.\mathsf{preid}}_{u} & \pi_u^i.\mathit{role} = \mathtt{resp} \end{cases}$$

$$\mathsf{clean}_{\mathtt{EL}}(u,i,[0]) = \begin{cases} \neg\pi_u^i.\mathsf{rev\_random}[0] \wedge \neg\mathsf{rev\_ltk}_{\pi_u^i.\mathsf{peerid}} & \pi_u^i.\mathit{role} = \mathtt{init} \\ \mathsf{clean}_{\mathtt{peerE}}(u,i,[0]) \wedge \neg\mathsf{rev\_ltk}_u & \pi_u^i.\mathit{role} = \mathtt{resp} \end{cases}$$

$$\mathsf{clean}_{\mathtt{EM}}(u,i,[0]) = \begin{cases} \neg\pi_u^i.\mathsf{rev\_random}[0] \wedge \neg\mathsf{rev\_mtk}^{\pi_u^i.\mathsf{peerpreid}}_{\pi_u^i.\mathsf{peerid}} & \pi_u^i.\mathit{role} = \mathtt{init} \\ \mathsf{clean}_{\mathtt{peerE}}(u,i,[0]) \wedge \neg\mathsf{rev\_mtk}^{\pi_u^i.\mathsf{preid}}_{u} & \pi_u^i.\mathit{role} = \mathtt{resp} \end{cases}$$

$$\mathsf{clean}_{\mathtt{EE}}(u,i,s,s') = \begin{cases} \neg\pi_u^i.\mathsf{rev\_random}[s] \wedge \mathsf{clean}_{\mathtt{peerE}}(u,i,s') & \pi_u^i.\mathit{role} = \mathtt{init} \\ \mathsf{clean}_{\mathtt{peerE}}(u,i,s) \wedge \neg\pi_u^i.\mathsf{rev\_random}[s'] & \pi_u^i.\mathit{role} = \mathtt{resp} \end{cases}$$

Within the same context as Definition 4, let $s_{ir} = [\mathbf{asym\text{-}ir}{:}x]$, $s_{ri} = [\mathbf{asym\text{-}ri}{:}x]$, $s'_{ir} = [\mathbf{asym\text{-}ir}{:}x-1]$ and $s'_{ri} = [\mathbf{asym\text{-}ri}{:}x-1]$. Define

$$\mathsf{clean}_{\mathtt{asym\text{-}ri}}(u,i,s_{ri}) = \begin{cases} \mathsf{clean}_{\mathtt{EE}}(u,i,[0],[0]) \vee \left(\mathsf{clean}_{\mathsf{state}}(u,i,s_{ri}) \wedge \mathsf{clean}_{\pi_u^i.\mathit{type}[0]}(u,i,[0])\right) & x = 1 \\ \mathsf{clean}_{\mathtt{EE}}(u,i,s'_{ri},s'_{ir}) \vee \left(\mathsf{clean}_{\mathsf{state}}(u,i,s_{ri}) \wedge \mathsf{clean}_{\mathtt{asym\text{-}ir}}(u,i,s'_{ir})\right) & x > 1 \end{cases}$$

$$\mathsf{clean}_{\mathtt{asym\text{-}ir}}(u,i,s_{ir}) = \begin{cases} \mathsf{clean}_{\mathtt{EE}}(u,i,s_{ri},[0]) \vee \left(\mathsf{clean}_{\mathsf{state}}(u,i,s_{ir}) \wedge \mathsf{clean}_{\mathtt{asym\text{-}ri}}(u,i,s_{ri})\right) & x = 1 \\ \mathsf{clean}_{\mathtt{EE}}(u,i,s_{ri},s'_{ir}) \vee \left(\mathsf{clean}_{\mathsf{state}}(u,i,s_{ir}) \wedge \mathsf{clean}_{\mathtt{asym\text{-}ri}}(u,i,s_{ri})\right) & x > 1 \end{cases}$$

Writing $s = [\mathbf{sym\text{-}ir}{:}x,y]$,

$$\mathsf{clean}_{\mathtt{sym\text{-}ir}}(u,i,s) = \mathsf{clean}_{\mathsf{state}}(u,i,s,s) \wedge \begin{cases} \mathsf{clean}_{\pi_u^i.\mathit{type}[0]}(u,i,[0]) & x = 0, y = 1 \\ \mathsf{clean}_{\mathtt{asym\text{-}ir}}(u,i,[\mathbf{asym\text{-}ir}{:}x]) & x > 0, y = 1 \\ \mathsf{clean}_{\mathtt{sym\text{-}ir}}(u,i,[\mathbf{sym\text{-}ir}{:}x,y-1]) & x \geq 0, y > 1 \end{cases}$$

There is no stage of type `sym-ri` with $x = 0$, so (writing now $s = [\mathbf{sym\text{-}ri}{:}x,y]$)

$$\mathsf{clean}_{\mathtt{sym\text{-}ri}}(u,i,s) = \mathsf{clean}_{\mathsf{state}}(u,i,s,s) \wedge \begin{cases} \mathsf{clean}_{\mathtt{asym\text{-}ri}}(u,i,[\mathbf{asym\text{-}ri}{:}x]) & x > 0, y = 1 \\ \mathsf{clean}_{\mathtt{sym\text{-}ri}}(u,i,[\mathbf{sym\text{-}ri}{:}x,y-1]) & x > 0, y > 1 \end{cases}$$

We may write $\mathsf{clean}_{\mathtt{sym}}$ to denote $\mathsf{clean}_{\mathtt{sym\text{-}ir}}$ or $\mathsf{clean}_{\mathtt{sym\text{-}ri}}$ where it is clear which one we mean.

Figure 4: Cleanness predicates. All notation is within the same context as Definition 4.

that all previous states in this symmetric chain are uncompromised, since later keys in the chain are computable from earlier states in the chain. Thinking recursively, this means that the previous stage's key derivation should have been secure, and that the adversary has not revealed the state linking the previous stage with the current one.

While the symmetric sending and receiving chains derive independent keys and are triggered differently during Signal protocol execution, their security properties are identical and captured by the following predicate; the different forms of the predicate are due to needing to properly name the "preceding" stage. There are different freshness conditions depending on whether the symmetric stage is used for a message from initiator to responder or vice versa. Moreover, the symmetric stages arising from the initial handshake ($x = 0$) and from subsequent asymmetric stages ($x > 0$) are subtly different.

5. Security Analysis

In this section we prove that Signal is a secure multi-stage key exchange protocol in the language of Section 4, under standard assumptions on the cryptographic building blocks.

The algorithms comprising the Signal protocol are given in Definition 1, and we summarise some key points below.

We have made a few minor reorganizations in Figure 1 compared to the actual implementation of Signal. We consider Signal to generate the first message keys for each chain at the same time that it initialises the chain, allowing us to consider these message keys as the session keys of the asymmetric stages. Similarly, we consider Bob to send his own one-time prekey eprepk_B instead of relaying it via the server. We mark these extra steps in Dark red in Figure 1.

KeyGen and MedTermKeyGen consist of uniform random sampling from the group.

Activate depends on the invoked role. Our prekey reorganization described above means that the roles of initiator and responder are technically reversed: although intuitively Alice initiates a session in our presentation, in fact Bob sends the first message, namely his prekeys (first right-to-left flow of Figure 1(b).

Thus, the activation algorithm for the responder (Bob) outputs a single one-time prekey and awaits a response. The activation algorithm for the initiator (Alice) outputs nothing and awaits incoming prekeys.

Run is the core protocol algorithm. It admits various cases, which we briefly describe. If the incoming message is the first, Run builds a session as described previously: for Alice, it operates as in the left side of Figure 1(b) and outputs a message containing epk_A; for Bob, it operates as in the right side of Figure 1(b) and outputs nothing.

After that, there are two cases: Run is either invoked to process an incoming message, or to encrypt an outgoing one. We distinguish between incoming ratchet public keys (causing asymmetric updates) and incoming messages (causing symmetric updates).

(i) *Outgoing message.* Perform a symmetric sending update, modifying the current sending chain key and using the resulting message key as the session key (left side of Figure 1(c)).

(ii) *Incoming ratchet public key.* If this ratchet public key has not been processed before, perform an asymmetric update using it to derive new sending and receiving chain keys as in Figure 1(d). Advance both chains by one step, and output the message keys as the session key for the two asymmetric sub-stages as indicated in the figure.

(iii) *Incoming message.* Use the message metadata to determine which receiving chain should be used for decryption, and which position the message takes in the chain. Advance that chain (according to the right side of Figure 1(c)) as many stages as necessary (possibly zero), storing for future use any message keys that were thus generated. Return as the session key the next receiving message key.

In the Signal protocol, old but unused receiving keys are stored at the peer for an implementation-dependent length of time, trading off forward security for transparent handling of outdated messages. This of course weakens the forward secrecy of the keys, though their other security properties remain the same. We choose not to model this weakened forward secrecy guarantee, passing only the latest chaining key from stage to stage.

With these definitions, we can consider the advantage of an adversary in a multi-stage key exchange security game against our model of the Signal protocol:

Theorem 1. *The Signal protocol is a secure multi-stage key exchange protocol under the GDH assumption and assuming all* KDF*s are random oracles. That is, if no efficient adversary can break the assumptions with non-negligible probability, then no efficient adversary can win the multi-stage key indistinguishability security experiment for Signal (and thereby distinguish any fresh message encryption key from random) with non-negligible probability.*

The definitions of the security assumptions and the proof appear in the full version [10].

6. Limitations

As a first analysis of a complex protocol, we have chosen (some) simplicity over a full analysis of all of Signal's features. We hope that our presentation and model can serve as a starting point for future analyses.

We discuss here some of the features included in Signal which we have explicitly chosen not to model and observe limitations of our results.

Protocol Components. *Non-Signal library components.* The open-source libraries contain various sections of code which are not considered part of the Signal protocol. For example, the "header encryption" variant of the Double Ratchet is used by Pond and included in the reference implementation, but not used by Signal itself. Likewise, there is support for online key exchanges instead of via the prekey server. As these components are not intended to be part of the Signal protocol, we do not analyse them.

Out-of-band key verification. To reduce the trust requirements on the prekey server, Signal supports a fingerprint mechanism for verifying public keys through an out-of-band channel. We simply assume that long-term and medium-term public key distribution is honest, and do not analyse the out-of-band verification channel.

Same key for Ed25519 signing and Curve25519 DH. Signal uses the same key *ik* for DH agreement and for signing the medium-term prekeys[6]. [15, 41] prove security of a similar scheme under the Gap-DH assumption, effectively showing that the signatures can be simulated using the hashing random oracle. We conjecture a similar argument could apply here, but do not prove it; instead, we omit the signatures from consideration and enforce authentication of the prekeys in the game. This enforced authentication means we do not capture the class of attacks in which the adversary corrupts an identity key and then inserts a malicious signed pre-key.

Out-of-order decryption. To decrypt out-of-order messages, users must store message keys until the messages arrive, reducing their forward security. As discussed in Section 5 we do not consider this storage.

Simultaneous session initiation. Signal has a mechanism to deal silently with the case that Alice and Bob simultaneously initiate a session with each other. Roughly, when an agent detects that this has happened they deterministically choose one party as the initiator (e.g. by sorting identity public keys and choosing the smaller), and then complete the session as if the other party had not acted. This requires a certain amount of trial and error: agents maintain multiple states for each peer, and attempt decryption of incoming messages in all of them. We do not consider this mechanism.

Other Security Goals and Threats. Our model describes key indistinguishability of two-party multi-stage key exchange protocols. There are other security

6. This is done in practise by reinterpreting the Curve25519 point as an Ed25519 key, and computing an EdDSA signature.

and functionality goals which Signal may address but which we do not study, including: group messaging properties[7], message sharing across multiple devices, voice and video call security, protocol efficiency (e.g. 0-round-trip modes), privacy, and deniability.

Implementation-specific threats. We make various assumptions on the components used by the protocol. In particular, we do not consider specific implementations of primitives (e.g. the particular choice of curve), instead assuming standard security properties. We also do not consider side-channel attacks.

Tightness of the security reduction. As pointed out in [1], a limitation of conventional game hopping proofs for AKE protocols is that they do not provide tight reductions. The underlying reason is that the reductions depend on guessing the specific party and session under attack. In the case of a widely deployed protocol with huge amounts of sessions, such as Signal, this leads to an extremely non-tight reduction. While [1] develops some new AKE protocols with tight reductions, their protocols are non-standard in their setup and assumptions. In particular, there is currently no known technique for constructing a tight reduction that is applicable to the Signal protocol.

Application Variants. Popular applications using Signal tend to change important details as they implement or integrate the protocol, and thus merit security analyses in their own right. For example, WhatsApp implements a re-transmission mechanism: if Bob appears to change his identity key, clients will resend messages encrypted under the new value. Hence, an adversary with control over identity registration can disconnect Bob and replace his key, and Alice will re-send the message to the adversary.

7. Conclusions and Future Work

In this work we provided the first formal security analysis of the cryptographic core of the Signal protocol. While any first analysis for such a complex object will be necessarily incomplete, our analysis leads to several observations.

First, our analysis shows that the cryptographic core of Signal provides useful security properties. These properties, while complex, are encoded in our security model, and which we prove that Signal satisfies under standard cryptographic assumptions. Practically speaking, they imply secrecy and authentication of the message keys which Signal derives, even under a variety of adversarial compromise scenarios such as forward security (and thus "future secrecy"). If used correctly, Signal could achieve a form of post-compromise security, which has substantial advantages over forward secrecy as described in [11].

Our analysis has also revealed many subtleties of Signal's security properties. For example, we identified six different security properties for message keys (`triple`, `triple+DHE`, `asym-ir`, `asym-ri`, `sym-ir` and `sym-ri`).

One can imagine strengthening the protocol further. For example, if the random number generator becomes fully predictable, it may be possible to compromise communications with future peers. We have pointed out to the developers that this can be solved at negligible cost by using constructions in the spirit of the NAXOS protocol [33] or including a static-static DH shared secret in the key derivation.

We have described some of the limitations of our approach in Section 6. Furthermore, the complexity and tendency to add "extra features" makes it hard to make statements about the protocol as it is used. Examples include the ability to reset the state [11], encrypt headers, or support out-of-order decryption.

As with many real-world security protocols, there are no detailed security goals specified for the protocol, so it is ultimately impossible to say if Signal achieves its goals. However, our analysis proves that several standard security properties are satisfied by the protocol, and we have found no major flaws in its design, which is very encouraging.

7. The implementation of group messaging is not specified at the protocol layer. If it is implemented using multiple pairwise sessions, its security may follow in a relatively straightforward fashion—however, there are many other possible security properties which might be desired, such as transcript consistency.

Acknowledgements

The authors acknowledge helpful discussions with Marc Fischlin and Felix Günther (TU Darmstadt) and valuable comments from Chris Brzuska (TU Hamburg) and Trevor Perrin (Open Whisper Systems).

References

[1] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. "Tightly-Secure Authenticated Key Exchange". In: *TCC 2015, Part I*. Vol. 9014. LNCS. Springer, Heidelberg, Mar. 2015, pp. 629–658.

[2] Chris Ballinger. *ChatSecure*. URL: https://chatsecure.org/blog/chatsecure-v4-released/ (visited on 01/2017).

[3] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. *Ratcheted Encryption and Key Exchange: The Security of Messaging*. Cryptology ePrint Archive, Report 2016/1028. http://eprint.iacr.org/2016/1028. 2016.

[4] Mihir Bellare and Bennet S. Yee. "Forward-Security in Private-Key Cryptography". In: *CT-RSA 2003*. Vol. 2612. LNCS. Springer, Heidelberg, Apr. 2003, pp. 1–18.

[5] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-Speed High-Security Signatures". In: *CHES 2011*. Vol. 6917. LNCS. Springer, Heidelberg, Sept. 2011, pp. 124–142.

[6] Karthikeyan Bhargavan, Christina Brzuska, Cédric Fournet, Matthew Green, Markulf Kohlweiss, and Santiago Zanella-Béguelin. "Downgrade Resilience in Key-Exchange Protocols". In: *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2016.

[7] David Bogado and Danny O'Brien. *Punished for a Paradox*. Mar. 2, 2016. URL: https://www.eff.org/deeplinks/2016/03/punished-for-paradox-brazils-facebook (visited on 07/2016).

[8] Nikita Borisov, Ian Goldberg, and Eric Brewer. "Off-the-record Communication, or, Why Not to Use PGP". In: WPES. Washington DC, USA: ACM, 2004, pp. 77–84.

[9] Ran Canetti, Shai Halevi, and Jonathan Katz. "A Forward-Secure Public-Key Encryption Scheme". In: *EUROCRYPT 2003*. Vol. 2656. LNCS. Springer, Heidelberg, May 2003, pp. 255–271.

[10] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. *A Formal Security Analysis of the Signal Messaging Protocol*. Cryptology ePrint Archive, Report 2016/1013. http://eprint.iacr.org/2016/1013. 2016.

[11] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. *On Post-Compromise Security*. (A shorter version of this paper appears at CSF 2016). 2016. URL: http://eprint.iacr.org/2016/221.

[12] *Conversations*. URL: https://conversations.im/ (visited on 07/2016).

[13] Cas Cremers and Michele Feltz. *One-round Strongly Secure Key Exchange with Perfect Forward Secrecy and Deniability*. Cryptology ePrint Archive, Report 2011/300. http://eprint.iacr.org/2011/300. 2011.

[14] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. "Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication". In: *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2016.

[15] Jean Paul Degabriele, Anja Lehmann, Kenneth G. Paterson, Nigel P. Smart, and Mario Strefler. "On the Joint Security of Encryption and Signature in EMV". In: *CT-RSA 2012*. Vol. 7178. LNCS. Springer, Heidelberg, Feb. 2012, pp. 116–135.

[16] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. "Deniable authentication and key exchange". In: *ACM CCS 06*. ACM Press, Oct. 2006, pp. 400–409.

[17] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. "Secure Off-the-record Messaging". In: WPES. Alexandria, VA, USA: ACM, 2005, pp. 81–89.

[18] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. "A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates". In: *ACM CCS 15*. ACM Press, Oct. 2015, pp. 1197–1210.

[19] Electronic Frontier Foundation. *Secure Messaging Scorecard*. 2016. URL: https://www.eff.org/node/82654.

[20] Facebook. *Messenger Secret Conversations*. Tech. rep. 2016. URL: https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf (visited on 07/2016).

[21] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Joerg Schwenk, and Thorsten Holz. *How Secure is TextSecure?* Cryptology ePrint Archive, Report 2014/904. http://eprint.iacr.org/2014/904 (Version from April 5, 2016). 2014.

[22] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Jörg Schwenk, and Thorsten Holz. "How Secure is TextSecure?" In: *1st IEEE European Symposium on Security and Privacy*. IEEE Computer Society Press, Mar. 2016.

[23] Christina Garman, Matthew Green, Gabriel Kaptchuk, Ian Miers, and Michael Rushanan. "Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage". In: *Usenix Security 2016*. 2016.

[24] Matthew D. Green and Ian Miers. "Forward Secure Asynchronous Messaging from Puncturable Encryption". In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 305–320.

[25] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. "On the Security of TLS-DHE in the Standard Model". In: *CRYPTO 2012*. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 273–293.

[26] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. "On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption". In: *ACM CCS 15*. ACM Press, Oct. 2015, pp. 1185–1196.

[27] Nadim Kobeissi. *Cryptocat*. URL: https://crypto.cat/security.html (visited on 07/2016).

[28] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. "Automated Verification for Secure Messaging Protocols and their Implementations: A Symbolic and Computational Approach". In: *2nd IEEE European Symposium on Security and Privacy*. IEEE Computer Society Press, Apr. 2017.

[29] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. "(De-)Constructing TLS 1.3". In: *INDOCRYPT 2015*. Vol. 9462. LNCS. Springer, Heidelberg, Dec. 2015, pp. 85–102.

[30] Hugo Krawczyk. "Cryptographic Extraction and Key Derivation: The HKDF Scheme". In: *CRYPTO 2010*. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 631–648.

[31] Hugo Krawczyk. "HMQV: A High-Performance Secure Diffie-Hellman Protocol". In: *CRYPTO 2005*. Vol. 3621. LNCS. Springer, Heidelberg, Aug. 2005, pp. 546–566.

[32] Caroline Kudla and Kenneth G. Paterson. "Modular Security Proofs for Key Agreement Protocols". In: *ASIACRYPT 2005*. Vol. 3788. LNCS. Springer, Heidelberg, Dec. 2005, pp. 549–565.

[33] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. "Stronger Security of Authenticated Key Exchange". In: *ProvSec 2007*. Vol. 4784. LNCS. Springer, Heidelberg, Nov. 2007, pp. 1–16.

[34] Adam Langley. *Pond*. 2014. URL: https://pond.imperialviolet.org/ (visited on 06/22/2015).

[35] Xinyu Li, Jing Xu, Zhenfeng Zhang, Dengguo Feng, and Honggang Hu. "Multiple Handshakes Security of TLS 1.3 Candidates". In: *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2016.

[36] `libsignal-protocol-java`. GitHub repository, commit hash `4a7bc1667a68c1d8e6af0151be30b84b94fd1e38`. 2016. URL: github.com/WhisperSystems/libsignal-protocol-java (visited on 07/2016).

[37] Moxie Marlinspike. *Advanced cryptographic ratcheting*. Blog. 2013. URL: https://whispersystems.org/blog/advanced-ratcheting/ (visited on 07/2016).

[38] Moxie Marlinspike. *Open Whisper Systems partners with Google on end-to-end encryption for Allo*. Blog. 2016. URL: https://whispersystems.org/blog/allo/ (visited on 07/2016).

[39] Alfred Menezes and Berkant Ustaoglu. "On Reusing Ephemeral Keys in Diffie–Hellman Key Agreement Protocols". In: *Int. J. Appl. Cryptol.* 2.2 (Jan. 2010), pp. 154–158.

[40] Vinnie Moscaritolo, Gary Belvin, and Phil Zimmermann. *Silent Circle Instant Messaging Protocol Specification*. Tech. rep. Archived from the original. Dec. 5, 2012. URL: https://web.archive.org/web/20150402122917/https://silentcircle.com/sites/default/themes/silentcircle/assets/downloads/SCIMP_paper.pdf (visited on 07/2016).

[41] Kenneth G. Paterson, Jacob C. N. Schuldt, Martijn Stam, and Susan Thomson. "On the Joint Security of Encryption and Signature, Revisited". In: *ASIACRYPT 2011*. Vol. 7073. LNCS. Springer, Heidelberg, Dec. 2011, pp. 161–178.

[42] Trevor Perrin. *Double Ratchet Algorithm*. GitHub wiki. 2016. URL: https://github.com/trevp/double_ratchet/wiki (visited on 07/22/2016).

[43] Trevor Perrin. *The XEdDSA and VXEdDSA Signature Schemes*. Specification. Oct. 2016. URL: https://whispersystems.org/docs/specifications/xeddsa/ (visited on 07/2016).

[44] Trevor Perrin and Moxie Marlinspike. *The Double Ratchet Algorithm*. Specification. Nov. 2016. URL: https://whispersystems.org/docs/specifications/doubleratchet/ (visited on 01/2017).

[45] Trevor Perrin and Moxie Marlinspike. *The X3DH Key Agreement Protocol*. Specification. Nov. 2016. URL: https://whispersystems.org/docs/specifications/x3dh/ (visited on 01/2017).

[46] J. Reardon, D. Basin, and S. Capkun. "SoK: Secure Data Deletion". In: *Security and Privacy (SP), 2013 IEEE Symposium on*. May 2013, pp. 301–315.

[47] Andreas Straub. *OMEMO Encryption*. Oct. 25, 2015. URL: https://conversations.im/xeps/multi-end.html (visited on 07/2016).

[48] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. "SoK: Secure Messaging". In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 232–249.

[49] Nik Unger and Ian Goldberg. "Deniable Key Exchanges for Secure Messaging". In: *ACM CCS 15*. ACM Press, Oct. 2015, pp. 1211–1223.

[50] WhatsApp. *WhatsApp Encryption Overview*. Tech. rep. 2016. URL: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf (visited on 07/2016).