A Formal Security Analysis of the Signal Messaging Protocol

Extended Version, November 2017[†]

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 [15] 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 [16, 29] 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 [36].

The Signal Protocol. While there has been a range of activity in end-to-end encryption for instant messaging [31, 68], 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" [53, 54]. 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 [54] 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

† *An overview of changes can be found in Appendix D. An extended abstract of this paper appears at IEEE EuroS&P 2017.*

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.

(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 [11]). 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 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 [55], WhatsApp [70], Facebook Messenger [32], as well as a host of variants in "secure messaging" apps, including Silent Circle [57], Pond [51], and (via the OMEMO extension [67] to XMPP) Cryptocat v2 [43], Conversations [24], and ChatSecure [4].

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 [14, 26, 30, 42, 46, 52].

Frosch et al. [34, 35] 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.

Providing a security analysis for the Signal protocol is challenging. 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 [23].

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 proof sketch in Section 5 and the full proof in Section B.3. Our full proof is in the random oracle model, but we have also outlined the steps required for a proof in the standard model as a delta to the original proof, using (a variant of) the PRF-ODH assumption. As our proof is essentially a case distinction, the latter addition is not only arguably using a more plausible cryptographic assumption, but also provides more concrete analysis of the different security guarantees depending on how a message key is derived in the Signal Protocol.

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 [20] 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 [16], have been claimed to offer properties such as "future secrecy". Future secrecy of protocols like Signal has been discussed in depth by

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.

Cohn-Gordon, Cremers, and Garratt [23]. 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. [10] develop generic security definitions for ratcheted key exchange in a different context, also based on a computational model with key indistinguishability. They describe a Diffie–Hellman based protocol that is somewhat similar to the Signal protocol in that it uses a ratcheting mechanism and updates state, and prove that it is secure in their model under an oracle Diffie–Hellman assumption. They also show how to combine symmetric encryption schemes with ratcheted key exchange schemes. Their model captures variations of "backward" (healing from compromise) and forward secrecy, but their model only allows for one-way communication between Alice and Bob, so the security notions are one-sided: if the receiver's long-term key is compromised then all security is lost. They also only capture the asymmetric type of ratcheting in this sense, and do not consider symmetric ratcheting. The authors explicitly identify modelling Signal as future work.

Also concurrently and published at EuroS&P 2017, Kobeissi et al. [44] use ProVerif and CryptoVerif to analyze a simplified variant of Signal specified in a JavaScript variant called ProScript. Note that their main focus is present a methodology for automated verification for secure messaging protocols and their implementations. Kobeissi et al. consider a variant of Signal (that, e.g., does not use symmetric ratcheting). They identify a possible key compromise impersonation (KCI) attack; we discuss this further in the context of our model in our discussion of freshness in Section 4.3. From the ProScript code, they automatically extract ProVerif models that consider a finite number of sessions without loops. The CryptoVerif models are created manually. In both cases, the analysis involves the systematic manual exploration of several combinations of compromised keys. In contrast, we set out to manually construct and prove the strongest possible property that holds for Signal. For the core protocol design, this allows us to prove a stronger and more-fine grained security property.

Recently, Green and Miers [38] 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, during registration or later. 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. 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 existing documentation but the ultimate authority was the implementation[2] [53]. After the time of writing, Open Whisper Systems published high-level specifications for X3DH [63] and the Double Ratchet [62] which help to clarify many details, although the codebase is still

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.

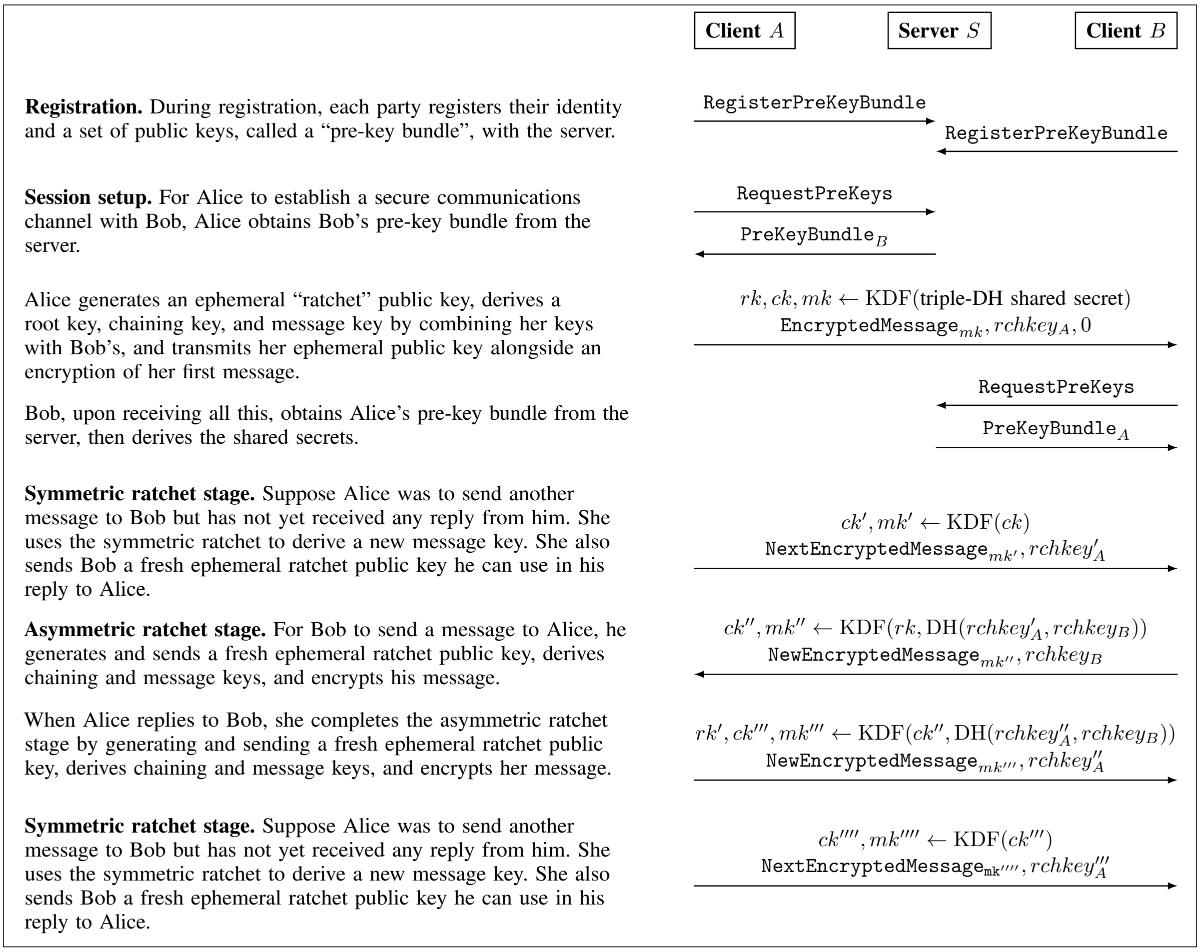


Figure 1: Message flow of an example Signal execution between two clients A and B via a server S. Notation and some operations have been simplified for clarity compared to later use.

necessary to obtain a full definition and the specification does not contain detailed definitions of the security goals.

2.1. Protocol Overview

Figure 1 shows the message flow for an example execution of the Signal protocol. The notation in Figure 1 has been simplified for clarity compared to later protocol diagrams; some operations have also been simplified, for example several KDF applications have been collapsed to give the general idea of the protocol.

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

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.

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.

Figure 4 on page 9 shows an example sequence of stages that one party might go through, with the message encryption keys derived in each stage. 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 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. At the end, we summarize the memory contents as used by Signal.

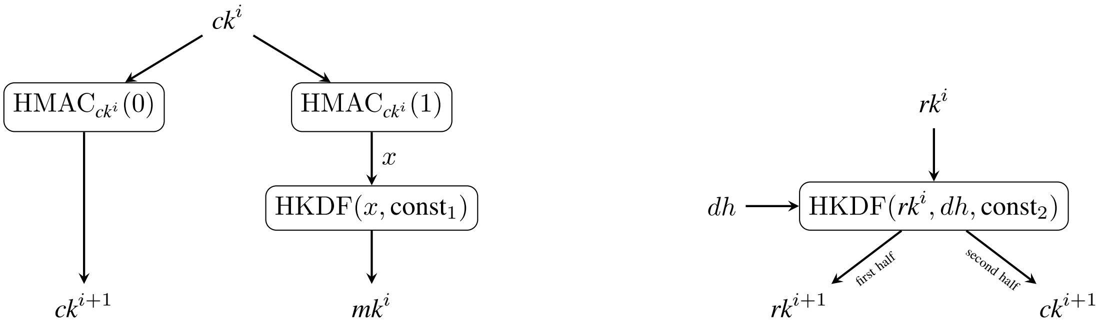
Table 1 is a glossary to the ten different classes of keys used in the Signal protocol, and Figure 2 shows the key derivations. Figure 3 depicts the operations executed in all stages of the protocol in a pseudocode format.

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 (Figure 4 on page 9). We adopt a unified notation to refer to any of them. All stages are described using a term in [square brackets]; the initial



(a) $\mathrm{KDF_m}$, the KDF for message chain updates. Note that new chain keys are *not* computed using HKDF; instead, they use only a HMAC.

(b) $\mathrm{KDF_r}$, the KDF for root chain updates. dh is the DH value derived for this stage update, and the result is a new root key as well an output chain key.

Figure 2: Key derivation functions for root and chain keys in Signal: keys flow along edges, and boxes apply their functions to their input. The diagrams depict the two functions used to take a chain key, and output a successor chain key and an exported key. In our analysis, we treat these functions as black boxes instead of making specific assumptions. Iterating these functions produces Signal's *chains*.

---

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).

There are two types of asymmetric updatesas part of the asymmetric ratchet; 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 $[\textbf{asym-ri}{:}x]$, and of the second type as $[\textbf{asym-ir}{:}x]$. 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 $[\textbf{asym-ri}{:}x]$ or $[\textbf{asym-ir}{:}x]$. 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 $[\textbf{sym-ri}{:}x{,}y]$ or $[\textbf{sym-ir}{:}x{,}y]$.

Signal accommodates the potential of out-of-order message delivery in the following way. 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. For example, the initiator performs all stages $[\textbf{sym-ir}{:}x{,}1], [\textbf{sym-ir}{:}x{,}2], \ldots$ before stage $[\textbf{asym-ir}{:}x + 1]$, but may delay stages $[\textbf{sym-ri}{:}x{,}y]$ as much as necessary. This means that Alice can retain any particular receiving chain for as long as she wants. Moreover, along any given chain, chain keys can be ratcheted forward to produce message keys in such a way that message keys are independent of each other so retaining them while waiting for late messages to arrive should not compromise other messages. This means that along a receiving chain, Alice can produce message key for delayed message 2, while still symmetrically ratcheting forward to decrypt received messages 3, 4, 5, etc., safe in the knowledge that retaining message key 2 while waiting for the message to arrive should not endanger other message keys along the chain. The two key concepts of the root key producing chain keys, and the chain keys producing message keys, means that messages can arrive in arbitrary order while Alice and Bob can continue to asymmetrically and symmetrically ratchet forward.

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^1 would be the root key derived by A in the initial stage, and $mk^{\textbf{sym-ri}:x,y}$ the message key derived in stage $[\textbf{sym-ri}{:}x{,}y]$. Not all stages derive all keys: for example, there is no $rk^{\textbf{sym-ri}:x,y}$, 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 (see Figure 4). 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 [12] or curve X448 [39]. The key derivation functions are depicted in Figure 2, anduse either $\mathrm{HMAC\text{-}SHA256}$ [6] or HKDF [47] using $\mathrm{SHA256}$as indicated.

AEAD denotes an authenticated encryption scheme with associated data [66]. 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. [35] 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 [13, 61]. Again, we treat it as a black-box signature.

2.3. Registration Phase—Figure 3(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 3(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 5, and then application of a key derivation function. While many possible variants of such protocols have been explored in-depth in the literature (HMQV [48], Kudla-Paterson [49], NAXOS [50] 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 [49].

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 [56] 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 ek_A, and computes a session key by performing three or four group exponentiations as depicted in Figure 5. She then concatenates the resulting shared secrets and passes them through a key derivation function (KDF_r,

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.

(a) Bob's registration phase (at install time), over an authentic channel **(Section 2.3)**

Client B | **Server** S

$ik_B, prek_B \xleftarrow{\$} \mathbb{Z}_q$

multiple $eprek_B \xleftarrow{\$} \mathbb{Z}_q$

$$\xrightarrow{ipk_B, prepk_B, \mathrm{Sign}_{ik_B}(prepk_B),\ \text{multiple } eprepk_B}$$

(b) Alice's session (Initiator) setup with peer Bob (Responder), over an authentic channel **(Section 2.4)**

Client instance π^i_A, stage $[0]$ | **Server** S

$$\xrightarrow{B}$$

$$\xleftarrow{ipk_B, prepk_B, \mathrm{Sign}_{ik_B}(prepk_B)[, eprepk_B]}$$

Client instance π^i_B, stage $[0]$

$ek_A \xleftarrow{\$} \mathbb{Z}_q$

$rchk^0_A \xleftarrow{\$} \mathbb{Z}_q$

$$\xrightarrow{epk_A,\ \text{key identifier for } prepk_B, rchpk^0_A[, eprepk_B]}$$

(in practice attached to initial encrypted message)

confirm possession of $prek_B[, eprek_B]$

$ms \leftarrow (prepk_B)^{ik_A} \| (ipk_B)^{ek_A} \| (prepk_B)^{ek_A}$

if $eprepk_B$ then $ms \leftarrow ms \| (eprepk_B)^{ek_A}$

$rk^1, ck^{\textbf{sym-ir}:0,0} \leftarrow \mathrm{KDF_r}(ms)$

$ck^{\textbf{sym-ir}:0,1}, mk^{\textbf{sym-ir}:0,0} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ir}:0,0})$

$ms \leftarrow (ipk_A)^{prek_B} \| (epk_A)^{ik_B} \| (epk_A)^{prek_B}$

if $eprepk_B$ then $ms \leftarrow ms \| (epk_A)^{eprek_B}$

$rk^1, ck^{\textbf{sym-ir}:0,0} \leftarrow \mathrm{KDF_r}(ms)$

$ck^{\textbf{sym-ir}:0,1}, mk^{\textbf{sym-ir}:0,0} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ir}:0,0})$

$rchk^0_B \xleftarrow{\$} \mathbb{Z}_q$

(c) Symmetric-ratchet communication: Alice sends a message to Bob **(Section 2.5)**

Client instance π^i_A, stage $[\textbf{sym-ir}{:}x{,}y]$ | Client instance π^i_B, stage $[\textbf{sym-ir}{:}x{,}y]$

$$\xrightarrow{\mathrm{AEAD}_{mk^{\textbf{sym-ir}:x,(y-1)}}(\text{message}, \text{AD} = \emptyset), rchpk^x_A, ipk_A, ipk_B, y}$$

(in practice $(rchpk^x_A, ipk_A, ipk_B, y)$ are included in the associated data of this message)

$ck^{\textbf{sym-ir}:x,y+1}, mk^{\textbf{sym-ir}:x,y} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ir}:x,y})$

$ck^{\textbf{sym-ir}:x,y+1}, mk^{\textbf{sym-ir}:x,y} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ir}:x,y})$

(d) Asymmetric-ratchet updates: Alice and Bob start new symmetric chains with new ratchet keys **(Section 2.6)**

Client π^i_A, stage $[\textbf{asym-ri}{:}x]$ | Client π^i_B, stage $[\textbf{asym-ri}{:}x]$

$tmp, ck^{\textbf{sym-ri}:x,0} \leftarrow \mathrm{KDF_r}(rk^x, (rchpk^{x-1}_A)^{rchk^{x-1}_B})$

$ck^{\textbf{sym-ri}:x,1}, mk^{\textbf{sym-ri}:x,0} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ri}:x,0})$

$$\xleftarrow{rchpk^{x-1}_B}$$

(in practice in the associated data of a later message encrypted with $mk^{\textbf{sym-ri}:x,0}_B$)

$tmp, ck^{\textbf{sym-ri}:x,0} \leftarrow \mathrm{KDF_r}(rk^x, (rchpk^{x-1}_B)^{rchk^{x-1}_A})$

$ck^{\textbf{sym-ri}:x,1}, mk^{\textbf{sym-ri}:x,0} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ri}:x,0})$

$rchk^x_A \xleftarrow{\$} \mathbb{Z}_q$

Client π^i_A, stage $[\textbf{asym-ir}{:}x]$ | Client π^i_B, stage $[\textbf{asym-ir}{:}x]$

$rk^{x+1}, ck^{\textbf{sym-ir}:x,0} \leftarrow \mathrm{KDF_r}(tmp, (rchpk^{x-1}_B)^{rchk^x_A})$

$ck^{\textbf{sym-ir}:x,1}, mk^{\textbf{sym-ir}:x,0} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ir}:x,0})$

$$\xrightarrow{rchpk^x_A}$$

(in practice in the associated data of a later message encrypted with $mk^{\textbf{sym-ir}:x,0}_A$)

$rk^{x+1}, ck^{\textbf{sym-ir}:x,0} \leftarrow \mathrm{KDF_r}(tmp, (rchpk^x_A)^{rchk^{x-1}_B})$

$ck^{\textbf{sym-ir}:x,1}, mk^{\textbf{sym-ir}:x,0} \leftarrow \mathrm{KDF_m}(ck^{\textbf{sym-ir}:x,0})$

$rchk^x_B \xleftarrow{\$} \mathbb{Z}_q$

Figure 3: 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, $\mathrm{KDF_r}$ and $\mathrm{KDF_m}$ in Figure 2, 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.

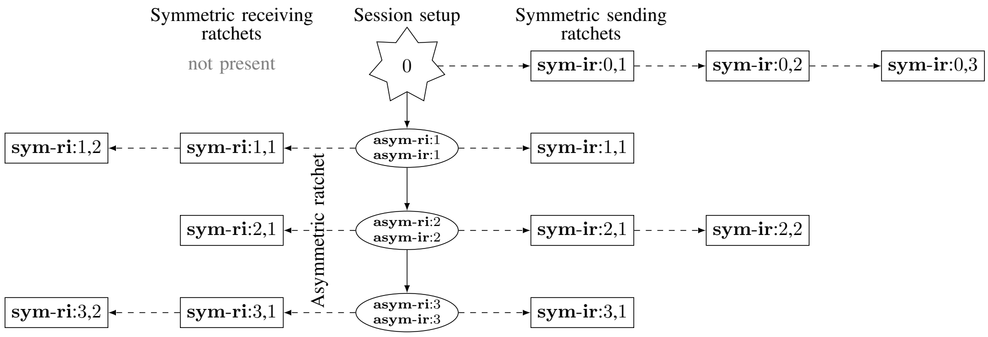


Figure 4: An example tree of *stages* that one party might use in one session of Signal. The content of each node is the stage name; recall that "ir" denotes stages deriving a key used to send from initiator to responder, and vice versa for "ri". In our notation, mk^s denotes the message key derived by stage s. For example, $mk^{\mathbf{sym\text{-}ir}:x,y}$ denotes the yth symmetric key on the xth chain, used by the initiator to encrypt messages and by the responder to decrypt them. Each chain is derived by ratcheting as in Figure 2 with a root or chain key. For the first symmetric ratchets in a session, the initiator of the session only has a sending chain, while the responder only has a receiving chain.

	initiator	responder	intended use
signed prekey	$prek_A$	$prek_B$	medium-term, reused across sessions
identity key	ik_A	ik_B	long-term, bound to identity
one-time (pre)key	ek_A	$eprek_B$	unique to each session, never reused

Figure 5: 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 14 will be partially based on this graph.

Figure 2b)to derive an initial root key rk^1 and sending chain key $ck^{\mathbf{sym\text{-}ir}:0,0}$. (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^1 and chain key (which he records as $ck^{\mathbf{sym\text{-}ir}:0,0}$). 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 $ck^{\mathbf{sym\text{-}ir}:0,1}$.

2.5. Symmetric-Ratchet Phase—Figure 3(c)

Two sequences of symmetric keys will be derived using a PRF chain, one for sending and one for receiving. The symmetric chains—to the left and the right in Figure 4—may 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

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.

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 3(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 3. 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 (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 3 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 3(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 3(d) will never be horizontally flipped.

2.7. Memory Contents

Signal is a stateful protocol, and a number of different values are available in Alice's memory at any time. Alice's global state—shared between all of her sessions—contains four different collections of values: identity keys (Alice's own identity private key, and the identity public keys of all her peers), signed prekeys, ephemeral keys, and a list of all sessions.

Furthermore, each session in the collection of sessions above contains the keys used by the protocol. Specifically, a session always has its agent's identity private key and its peer's identity public key, a current root and sending chain key, and a current ratchet key. In addition, it has some number of receiving chain and message keys, corresponding to out-of-order messages not yet received from the peer.

3. Threat Models

We will analyse 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 [23]). 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 14, 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 [53] states that Signal "is a ratcheting forward secrecy protocol that works in synchronous and asynchronous messaging environments". A separate GitHub wiki page

[60] 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. Parallels can be drawn, for example, with the TLS 1.3 standard [65, Appendix D], which discusses the following properties (where the network is fully adversarially-controlled, and where the adversary may compromise the keys of some participants).

Correctness If Alice and Bob complete an exchange together then they should derive the same keys; distinct exchanges should derive distinct keys.

Secrecy If Alice and Bob complete an exchange to generate a key k, nobody other than Alice and Bob should be able to learn anything about k.

(Implicit) Authentication If Alice believes that she shares the key k with Bob, nobody other than Bob should be able to learn anything about k. Note that this property is implied by secrecy.

Forward secrecy An attacker who compromises Alice's long-term secret after a session is complete should not be able to learn anything about the keys derived in that session.

Identity hiding A passive adversary should not learn the identity of partners to a session.

It is common in the authenticated key exchange literature to assume a trusted public key infrastructure (PKI), though some models allow the adversary more control [17]. In Signal the PKI is combined with the network, in the sense that the same server distributes identity and ephemeral keys. Thus, in some sense assuming a trusted PKI also restricts the attacker's control over particular sessions.

Some claims have been made about privacy and deniability [69] 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 [28, full deniability] to *someone* but perhaps not to *Bob* [25, 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 [64], 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 3 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 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* [7, 8] (albeit a multi-stage variant), as opposed to a monolithic *secure channel* notion such as authenticated and confidential channel establishment (ACCE) [41]. It is often preferable to analyze the key exchange portion independently from the message transport layer, and then compose this with authenticated encryption to establish a secure channel [21]. Monolithic notions like ACCE are necessary for protocols such as TLS, which use the session key (or values derived from it) in the channel establishment and thus prevent a clean separation for composition. As indicated by the parenthetical comments under message arrows in Figure 3, Signal uses message keys to authenticate not just data (which is omitted from our key exchange model) but also handshake messages. In our proof, we therefore modify the protocol to instead send these handshake messages

5. Future secrecy means "a leak of keys to a passive eavesdropper will be healed by introducing new DH ratchet keys" [60].

in the clear, with authentication enforced via our freshness predicate. We discuss these modifications further in Section B.1.

Another subtlety compared to the multi-stage key exchange model of Fischlin and Günther is that QUIC and TLS 1.3 demand a linear sequence of stages, whereas for our model of Signal we use a tree of stages, as seen in Figure 4.

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; $[\mathbf{sym\text{-}ir}{:}x{,}y]$ or $[\mathbf{sym\text{-}ri}{:}x{,}y]$ for symmetric sending or receiving stages; or $[\mathbf{asym\text{-}ri}{:}x]$ or $[\mathbf{asym\text{-}ir}{:}x]$ for the 1st and 2nd portions of the xth asymmetric stage. (See also Figure 4 andFigure 3.)

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".

Generality of our model. Some aspects of our model are quite general, and others are very specific to Signal. Our formulation of a multi-stage key exchange protocol as a tuple of algorithms, as well as the main experiment and oracles in Figure 6, should be applicable to any multi-stage key exchange protocol that includes semi-static (medium term) keys. However, our freshness definition is highly customized to Signal via our clean clauses, since we aim to precisely characterize the security properties of Signal's keys.

The level of generality is an important decision when designing a security model for a protocol like Signal: on one hand, a general model allows analysis of and comparison to other protocols; on the other, of necessity it does not allow fine-grained statements about the specific protocol under consideration. Our model lies towards the centre of this spectrum: we aim to keep the overall structure relatively independent of Signal (though of necessity we added support for medium-term keys), while the cleanness predicates described later allow us to make fine-grained assertions which capture as much as possible.

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 [27, 59], 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:

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

- $\pi.\mathit{role} \in \{\texttt{init}, \texttt{resp}\}$: the instance's role
- $\pi.\mathit{peeripk}$: the peer's long-term public key

- π.*peerprepk*: the peer's medium-term public key
- π.*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
- π.*st*$[s]$: any additional protocol state values that a previous stage gives as input to stage s (defined as part of the protocol).
- π.*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 3.
- π.*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 a multi-stage key exchange protocol, we can now set up 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 6. 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.

Note that $\mathsf{Exp}^{\mathsf{ms\text{-}ind}}$ includes the following global variables:
- b: a challenge bit
- $\mathsf{tested} = (u, i, s)$ or $\perp$: recording the inputs to the query $\mathsf{Test}(u, i, s)$ or $\perp$ if no Test query has happened

Furthermore, $\mathsf{Exp}^{\mathsf{ms\text{-}ind}}$ extends the per-session state π^i_u with the following experiment variables:
- $\pi^i_u.\mathsf{rand}[s] \in \{0,1\}^\lambda$: random coins for π^i_u's sth stage
- $\pi^i_u.\mathsf{peerid} \in \{1, \ldots, \mathsf{n_P}\}$: the identifier of the alleged peer
- $\pi^i_u.\mathsf{peerpreid} \in \{1, \ldots, \mathsf{n_M}\}$: the index of the alleged peer's medium-term key
- $\pi^i_u.\mathsf{rev\_session}[s] \in \{\mathrm{true}, \mathrm{false}\}$: whether $\mathsf{RevSessKey}(u, i, s)$ was called or not; default false
- $\pi^i_u.\mathsf{rev\_random}[s] \in \{\mathrm{true}, \mathrm{false}\}$: whether $\mathsf{RevRand}(u, i, s)$ was called or not; default false
- $\pi^i_u.\mathsf{rev\_state}[s] \in \{\mathrm{true}, \mathrm{false}\}$: whether $\mathsf{RevState}(u, i, s)$ was called or not; default false

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 π^i_u.*peeripk* being set; and similarly for the semi-static key index $\pi^i_u.\mathsf{peerpreid}$ upon π^i_u.*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

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

$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$\mathsf{tested} \leftarrow \perp$
// generate long-term and semi-static keys
for $u = 1$ **to** $\mathsf{n_P}$ **do**
  $(ipk_u, ik_u) \xleftarrow{\$} \mathrm{KeyGen}()$
  for $\mathsf{preid} = 1$ **to** $\mathsf{n_M}$ **do**
    $(prepk_u^{\mathsf{preid}}, prek_u^{\mathsf{preid}}) \xleftarrow{\$} \mathrm{MedTermKeyGen}(ik_u)$
$pubinfo \leftarrow (ipk_1, \ldots, ipk_{\mathsf{np}}, prepk_1^1, \ldots, prepk_{\mathsf{np}}^{\mathsf{n_M}})$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathsf{Send},\mathsf{Rev}*,\mathsf{Test}}(pubinfo)$
if $(\mathsf{tested} \neq \perp) \wedge \mathsf{fresh}(\mathsf{tested}) \wedge \mathsf{b} = \mathsf{b}'$ **then**
  return 1 *// the adversary wins*
else
  return 0 *// the adversary loses*

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

if $\pi_u^i = \perp$ **then**
  // start new session and record intended peer
  parse m as $(\pi_u^i.\mathsf{preid}, role)$
  $\pi_u^i.\vec{\mathsf{rand}} \xleftarrow{\$} \{0,1\}^{\mathsf{n_s} \times \lambda}$
  $(\pi_u^i, m') \leftarrow \mathrm{Activate}(ik_u, prek_u^{\pi_u^i.\mathsf{preid}}, role; \pi_u^i.\mathsf{rand}[0])$
  return m'
else
  $s \leftarrow \pi_u^i.stage$
  $(\pi_u^i, m') \leftarrow \mathrm{Run}(ik_u, prek_u^{\pi_u^i.\mathsf{preid}}, \pi_u^i, m; \pi_u^i.\mathsf{rand}[s])$
  return m'

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

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

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

$\mathsf{rev\_ltk}_u \leftarrow \mathrm{true}$
return ik_u

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

$\mathsf{rev\_mtk}_u^{\mathsf{preid}} \leftarrow \mathrm{true}$
return $prek_u^{\mathsf{preid}}$

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

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

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

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

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

// can only call Test *once, and only on accepted stages*
if $(\mathsf{tested} \neq \perp)$ **or** $(\pi_u^i.status[s] \neq \texttt{accept})$ **then**
  return $\perp$
$\mathsf{tested} \leftarrow (u, i, s)$
// return real or random key depending on b
if $\mathsf{b} = 0$ **then**
  return $\pi_u^i.k[s]$
else
  $k' \xleftarrow{\$} \mathcal{K}$
  return k'

Figure 6: Security experiment for adversary $\mathcal{A}$ against multi-stage key indistinguishability security of protocol Π.

derivation or associated data of encrypted messages. This means that the unknown key share attack against TextSecure [35] 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 when defining fresh 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. That is, we use the structure of the protocol to infer which attacks cannot possibly be prevented, and rule them out by restricting the adversary. Our goal is 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).

We must define fresh separately for the initial stages and for subsequent ones, since additional secrets are introduced in later asymmetric stages. In the initial stages, our choices are based on Figure 5 on page 9. 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.

A vertex cover[6] of Figure 5 gives a way for the adversary to compute the relevant session key directly. We can think of our freshness predicate as excluding all such vertex covers; if all DH pairs were included in the KDF, this would yield the standard eCK freshness predicate.

In stages after the initial ones, we define modified freshness conditions to capture Signal's post-compromise security properties. These conditions are recursive: either the stage was fresh already, or it becomes fresh through the introduction of new secrets.

6. A vertex cover of a graph is a set of nodes incident to every edge.

name	sid	
sid[0]	$(\texttt{triple} : \mathit{ipk}_i, \mathit{ipk}_r, \mathit{prepk}_r, \mathit{epk}_i)$	if $\mathit{type}[0] = \texttt{triple}$
	$(\texttt{triple+DHE} : \mathit{ipk}_i, \mathit{ipk}_r, \mathit{prepk}_r, \mathit{epk}_i, \mathit{eprepk}_r)$	if $\mathit{type}[0] = \texttt{triple+DHE}$
$\mathit{sid}[\textbf{asym-ri}{:}x]$	$\mathit{sid}[0] \parallel (\texttt{asym-ri} : \mathit{rchpk}_i^0, \mathit{rchpk}_r^0)$	if $x = 1$
	$\mathit{sid}[\textbf{asym-ir}{:}x-1] \parallel (\texttt{asym-ri} : \mathit{rchpk}_r^{x-1})$	if $x > 1$
$\mathit{sid}[\textbf{asym-ir}{:}x]$	$\mathit{sid}[\textbf{asym-ri}{:}x] \parallel (\texttt{asym-ir} : \mathit{rchpk}_i^x)$	if $x > 0$
$\mathit{sid}[\textbf{sym-ri}{:}x,y]$	does not exist	if $x = 0$
	$\mathit{sid}[\textbf{asym-ri}{:}x] \parallel (\texttt{sym-ri} : y)$	if $x > 0$
$\mathit{sid}[\textbf{sym-ir}{:}x,y]$	$\mathit{sid}[0] \parallel (\texttt{sym} : y)$	if $x = 0$
	$\mathit{sid}[\textbf{asym-ir}{:}x] \parallel (\texttt{sym-ir} : y)$	if $x > 0$

TABLE 2: Definition of session identifiers $\mathit{sid}[s]$ 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 $\mathit{sid}[0]$. We note that $\mathit{sid}[\textbf{sym-ir}{:}x,y]$ for $x = 0$ does not exist because the receiver never starts a symmetric chain immediately after the handshake, always first performing a DH ratchet.

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

Definition 4 (Validity and freshness)**.** Let s be the i^{th} session at agent u, and let $\tau = \pi_u^i.\mathit{type}[s]$ be its type (e.g. `triple`, `triple+DHE`, …). It is is valid if it has accepted and the adversary has not revealed either its session key or the session key of any session with the same identifier, and fresh if it additionally satisfies cleanness:

$$\begin{aligned}
\mathsf{clean}_\tau(u,i,s) &\quad \text{is defined in the following sections} \\
\mathsf{valid}(u,i,s) &= (\pi_u^i.\mathit{status}[s] = \texttt{accept}) \wedge \neg\pi_u^i.\mathsf{rev_session}[s] \\
&\quad \wedge \left(\forall\, j : \pi_u^i.\mathit{sid}[s] = \pi^j_{\pi_u^i.\mathsf{peerid}}.\mathit{sid}[s] \implies \neg\pi^j_{\pi_u^i.\mathsf{peerid}}.\mathsf{rev_session}[s]\right) \\
\mathsf{fresh}(u,i,s) &= \mathsf{valid}(u,i,s) \wedge \mathsf{clean}_\tau(u,i,s)
\end{aligned}$$

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]**.** The session key derived from a `triple` (resp. `triple+DHE`) key exchange is derived from the concatenation of three (resp. four) DH shared secrets; thus, it will be secret as long as at least one of the input secrets is. The cleanness predicate in this stage is thus the disjunction of three predicates, each encoding the secrecy of one DH pair. Note that $\mathsf{clean}_{\texttt{triple}}$ and $\mathsf{clean}_{\texttt{triple+DHE}}$ only need to be defined for the initial key exchange, i.e., stage [0].

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

$$\mathsf{clean}_{\texttt{triple}}(u,i,[0]) = \mathsf{clean}_{\texttt{LM}}(u,i) \vee \mathsf{clean}_{\texttt{EL}}(u,i,0) \vee \mathsf{clean}_{\texttt{EM}}(u,i,0)$$

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

$$\mathsf{clean}_{\texttt{triple+DHE}}(u,i,[0]) = \mathsf{clean}_{\texttt{triple}}(u,i,[0]) \vee \mathsf{clean}_{\texttt{EE}}(u,i,0,0)$$

For the sub-clauses $\mathsf{clean}_{\texttt{XY}}$ 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 5. This necessitates the two definitions below of $\mathsf{clean}_{\texttt{LM}}/\mathsf{clean}_{\texttt{EL}}/\mathsf{clean}_{\texttt{EM}}$ 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 first that it was actually generated by the intended peer (meaning that the peer session must exist), and second that it was not subsequently revealed (identifying the peer session using session identifiers). The following clause helps identify that precise situation:

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

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 will succeed.

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

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

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

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

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.

Excluded attacks. Recall that a vertex cover of Figure 5 on page 9 gives an attack which we rule out. Any cover must include one of prek_B and ek_A to meet the edge between them, so the only (minimal) vertex covers for a `triple` handshake are the full state of B ($\mathit{prek}_B, \mathit{ik}_B$), the full state of A (ek_A, ik_A), or the pair (prek_B, ek_A). The former two are trivial: an agent must be able to compute its own session key, so learning all their secrets also allows the adversary to compute it. The final pair exists because of the lack of an edge in Signal between ik_A and ik_B, and means that an adversary who learns prek_B and ek_A can learn the session key. In particular, since the ephemeral key is not authenticated, the adversary can generate their own ek_A and successfully impersonate A. This is the key compromise impersonation (KCI) attack of [44]; it is ruled out in our model because $\mathsf{clean}_{\mathtt{triple}}$ is false if both prek_B and ek_A have been revealed.

Since a vertex cover for a `triple+DHE` handshake must be a superset of the above, the only non-trivial one is again (prek_B, ek_A); this means that the KCI attack succeeds even against a `triple+DHE` handshake.

4.3.2. Asymmetric Stages[asym-ir:x]/[asym-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

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

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[**asym-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[**asym-ir**:x] and should be secure if at least one of the three pieces is unrevealed, which is what `asym-ir` captures.

Definition 8 ($\mathsf{clean}_{\texttt{asym-ir}}, \mathsf{clean}_{\texttt{asym-ri}}$)**.** Within the same context as Definition 4, let $s_{ir} = [\textbf{asym-ir}{:}x]$, $s_{ri} = [\textbf{asym-ri}{:}x]$, $s'_{ir} = [\textbf{asym-ir}{:}x-1]$ and $s'_{ri} = [\textbf{asym-ri}{:}x-1]$. Define

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

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

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 keys are compromised, and thus the second disjuncts are not 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 session is clean. Similarly, vice versa the session can be clean even if the current ephemeral exchange is compromised, just so long as the previous prior secrets are uncompromised. 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 $[\textbf{sym-ir}{:}x{,}y]$ **and** $[\textbf{sym-ri}{:}x{,}y]$**.** For stages with only symmetric ratcheting, new session keys should be secure only if the state is unknown to the adversary: this demands 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.

Definition 9 ($\mathsf{clean}_{\mathtt{sym}}$)**.** Writing $s = [\textbf{sym-ir}{:}x{,}y]$,

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

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

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

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

Excluded attacks. Since no additional secrets are included in message keys derived from symmetric ratchet stages, these predicates simply require that the adversary has not compromised any previous state along the chain: neither the asymmetric stage which created the chain, nor any of the intermediate symmetric stages, are permitted targets for queries. In other words, we exclude the attack in which the adversary corrupts a chain key and computes subsequent messages keys from it.

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 3 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 3.

$\underline{\mathrm{KeyGen}}$ and $\underline{\mathrm{MedTermKeyGen}}$ 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 3(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 3(b) and outputs a message containing epk_A; for Bob, it operates as in the right side of Figure 3(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 3(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 3(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 3(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.*

Proof (sketch). We give here a proof sketch. The full details and definitions of the security assumptions can be found in the Appendix. The proof considers of each stage type and sub-clause exhaustively, and is structured using the sequence-of-games technique.

Stage 0. We start by proving the security of the stage 0 key that is output by the `triple` key-exchange during session setup. We show this via taking cases over the disjuncts in the $\mathsf{clean}_{\mathtt{triple}}$ clause—over the different ways the session could be clean—noting that one of $\mathsf{clean}_{\mathtt{LM}}(u, i)$, $\mathsf{clean}_{\mathtt{EL}}(u, i, 0)$, $\mathsf{clean}_{\mathtt{EM}}(u, i, 0)$ must be upheld.

We bound each of these probabilities in turn by the advantage of reduction algorithms to the security experiments of our primitives—to DH security using the GDH and ROM assumptions.

Asymmetric stages. Next we consider the security of a stage s key such that stage s has stage type `asym-ir` or `asym-ri`. Again, we take cases over the different ways to satisfy the cleanness predicate, depending on the type of the stage. Most cases are of the form $\mathsf{clean}_{\mathtt{EE}}$, and for these we obtain a probability bound by replacing the DH ratchet keys and shared secrets with values from a GDH challenger.

The only case not of this form involves $\mathsf{clean}_{\mathsf{state}}$, which describes a scenario where both recent ratchet keys were compromised but the previous stage was still secure. Secrecy here is intuitive, and the bound follows from an inductive argument: if an adversary could win in this manner, then, assuming GDH and ROM security, there is an adversary which could win against the previous stage.

Symmetric stages. Finally, we consider the security of stage s keys of type `sym`. Here there is no disjunction in the cleanness predicate and hence only one case to consider. We replace the keys used to initialise the current sending chain with uniformly random values, since an adversary who could detect this could win against that previous stage.

Conclusion. The theorem follows by summing probabilities. □

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[7]. [27, 59] 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 and functionality goals which Signal may address but which we do not study, including: group messaging properties[8], 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 [2], 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 [2] 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

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

8. 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.

(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 [23].

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 [50] 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 [23], encrypt headers, or support out-of-order decryption. Cohn-Gordon and Cremers [22] discuss these limitations in more detail.

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.

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] Indrajit Ray, Ninghui Li, and Christopher Kruegel: eds. *ACM CCS 15*. ACM Press, Oct. 2015.

[2] 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.

[3] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. *On the Impossibility of Tight Cryptographic Reductions*. Cryptology ePrint Archive, Report 2015/374. http://eprint.iacr.org/2015/374. 2015.

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

[5] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. “An uninstantiable random-oracle-model scheme for a hybrid-encryption problem”. In: *Advances in Cryptology-EUROCRYPT 2004*. Springer. 2004, pp. 171–188.

[6] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. “A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols (Extended Abstract)”. In: *30th ACM STOC*. ACM Press, May 1998, pp. 419–428.

[7] Mihir Bellare, David Pointcheval, and Phillip Rogaway. “Authenticated Key Exchange Secure against Dictionary Attacks”. In: *EUROCRYPT 2000*. Vol. 1807. LNCS. Springer, Heidelberg, May 2000, pp. 139–155.

[8] Mihir Bellare and Phillip Rogaway. “Entity Authentication and Key Distribution”. In: *CRYPTO’93*. Vol. 773. LNCS. Springer, Heidelberg, Aug. 1994, pp. 232–249.

[9] Mihir Bellare and Phillip Rogaway. “Random oracles are practical: A paradigm for designing efficient protocols”. In: *Proceedings of the 1st ACM conference on Computer and communications security*. ACM. 1993, pp. 62–73.

[10] 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.

[11] 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.

[12] Daniel J. Bernstein. “Curve25519: New Diffie-Hellman Speed Records”. In: *PKC 2006*. Vol. 3958. LNCS. Springer, Heidelberg, Apr. 2006, pp. 207–228.

[13] 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.

[14] 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.

[15] 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).

[16] 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.

[17] Colin Boyd, Cas Cremers, Michele Feltz, Kenneth G. Paterson, Bertram Poettering, and Douglas Stebila. “ASICS: Authenticated Key Exchange Security Incorporating Certification Systems”. In: *ESORICS 2013*. Vol. 8134. LNCS. Springer, Heidelberg, Sept. 2013, pp. 381–399.

[18] Jacqueline Brendel, Marc Fischlin, Felix Günther, and Christian Janson. “PRF-ODH: Relations, Instantiations, and Impossibility Results”. In: *CRYPTO 2017, Part III*. Vol. 10403. LNCS. Springer, Heidelberg, Aug. 2017, pp. 651–681.

[19] Ran Canetti, Oded Goldreich, and Shai Halevi. “The random oracle methodology, revisited”. In: *Journal of the ACM (JACM)* 51.4 (2004), pp. 557–594.

[20] 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.

[21] Ran Canetti and Hugo Krawczyk. “Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels”. In: *EUROCRYPT 2001*. Vol. 2045. LNCS. Springer, Heidelberg, May 2001, pp. 453–474.

[22] Katriel Cohn-Gordon and Cas Cremers. *Mind the Gap: Where Provable Security and Real-World Messaging Don’t Quite Meet*. Cryptology ePrint Archive, Report 2017/982. http://eprint.iacr.org/2017/982. 2017.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

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

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

[30] 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.

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

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

[33] Marc Fischlin and Felix Günther. "Multi-Stage Key Exchange and the Case of Google's QUIC Protocol". In: *ACM CCS 14*. ACM Press, Nov. 2014, pp. 1193–1204.

[34] 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.

[35] 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.

[36] 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.

[37] Shafi Goldwasser and Yael Tauman Kalai. "Cryptographic Assumptions: A Position Paper". In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 907.

[38] 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.

[39] Mike Hamburg. *Ed448-Goldilocks, a new elliptic curve*. Cryptology ePrint Archive, Report 2015/625. http://eprint.iacr.org/2015/625. 2015.

[40] *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015.

[41] 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.

[42] 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.

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

[44] 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.

[45] Neal Koblitz and Alfred J Menezes. "The random oracle model: A twenty-year retrospective". In: *Designs, Codes and Cryptography* 77.2-3 (2015), pp. 587–610.

[46] 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.

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

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

[49] 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.

[50] 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.

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

[52] 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.

[53] `libsignal-protocol-java`. GitHub repository, commit hash `94fd1e38`. 2016. URL: github.com/WhisperSystems/libsignal-protocol-java (visited on 07/2016).

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

[55] 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).

[56] 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.

[57] 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).

[58] Tatsuaki Okamoto and David Pointcheval. "The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes". In: *PKC 2001*. Vol. 1992. LNCS. Springer, Heidelberg, Feb. 2001, pp. 104–118.

[59] 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.

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

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

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

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

[64] 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.
[65] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Internet-Draft draft-ietf-tls-tls13-14. July 2016. URL: http://www.ietf.org/internet-drafts/draft-ietf-tls-tls13-14.txt.
[66] Phillip Rogaway. “Authenticated-Encryption With Associated-Data”. In: *ACM CCS 02*. ACM Press, Nov. 2002, pp. 98–107.
[67] Andreas Straub. *OMEMO Encryption*. Oct. 25, 2015. URL: https://conversations.im/xeps/multi-end.html (visited on 07/2016).
[68] 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.
[69] Nik Unger and Ian Goldberg. “Deniable Key Exchanges for Secure Messaging”. In: *ACM CCS 15*. ACM Press, Oct. 2015, pp. 1211–1223.
[70] WhatsApp. *WhatsApp Encryption Overview*. Tech. rep. 2016. URL: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf (visited on 07/2016).

Appendix A.
On Hardness Assumptions and the Random Oracle Model (ROM)

When performing a game-hopping security proof in an extended Canetti-Krawczyk-style model, after each hop we must show that the resulting game is similar to the original. If certain values have been changed, the queries whose results differ must be simulated in an indistinguishable manner.

In particular, the eCK family of models all contain a query $\mathsf{RevSessKey}$ which reveals the session key derived by a targeted session. This models for example cryptanalysis of a large volume of encrypted traffic, or the ability to read certain locations in memory. When replacing certain DH keys with random values, we must ensure that the resulting game is similar to its original. For protocols using only ephemeral DH values g^x and g^y to compute session keys from g^{xy}, replacing g^x and g^y by random does not affect other sessions, and thus other $\mathsf{RevSessKey}$ queries are not affected. However, for more complex protocols (such as NAXOS and HMQV) in which the long-term keys are also included in the session key derivation, this game hop becomes more complex. Specifically, when the long-term keys are modified, *all* $\mathsf{RevSessKey}$ queries are affected, and their simulation is no longer trivial.

There is a proof obligation to show that the simulation of these queries does not allow an adversary to distinguish the two games. One way to do this is by using Gap-DH in the random oracle model, assuming that the KDF is a random oracle. In the simulation, whenever the adversary makes a query to the random oracle, the challenger tests the relevant part of the argument using the DDH oracle to determine whether the adversary has successfully derived the DH secret. If so, the simulation can terminate and the challenger uses this value in the Gap-DH game. This is the approach we take.

There are known issues with the ROM. An alternative to Gap-DH is to take a $\mathsf{PRF\text{-}ODH}$ (pseudorandom function with oracle DH) assumption, which effectively provides an oracle for session key computations, and (roughly) asserts that it is hard to solve computational DH even with access to the oracle. The game hop then takes the computational DH values from the $\mathsf{PRF\text{-}ODH}$ game, and answers $\mathsf{RevSessKey}$ queries by querying the oracle. The probability jump over the game is thus bounded by the $\mathsf{PRF\text{-}ODH}$ advantage.

There is a further complication in the case of Signal. In most normal DH protocols, there is only one method to compute a session key given a collection of secret inputs; such a method could be called a "combinator". For example, in NAXOS the combinator hashes one long-term key and uses that as a DH exponential. In Signal, on the other hand, there are many different combinators, and the oracle we use must be sufficiently flexible to simulate all of them. Thus, we have the following options:

(i) Define a $\mathsf{PRF\text{-}ODH}$ game parameterised by the combinator K used to assemble secrets into the arguments to the KDF. For each different type of key in Signal, assume hardness of this game and use this assumption to justify a game hop.
(ii) Assume that the KDF is a random oracle, and justify the game hop directly from the ROM and Gap-DH.

We choose the latter option, since we believe that the former hardness assumption is not necessarily justified. However, we conjecture that a carefully-formulated $\mathsf{PRF\text{-}ODH}$ game could be proven hard in the ROM, and therefore that one proof could effectively take either option depending on the reader's opinions. We leave such a game for future work.

Definitions of Hardness Assumptions

Our proof of security relies on standard cryptographic hardness assumptions related to DH key exchange. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order q generated by g, let $\alpha, \beta, \gamma \xleftarrow{\$} \mathbb{Z}_q$, and let $\mathcal{O}_{\text{DDH}}$ be an efficient black box algorithm (oracle) that, on input (g^x, g^y, g^z), outputs 1 if $g^z = g^{xy}$ and 0 otherwise. For any algorithm $\mathcal{D}$ let

$$\begin{aligned}\epsilon_{\text{DDH}}(\mathcal{D}) &:= \Big| \Pr\left[\mathcal{D}(\mathbb{G}, q, g, g^\alpha, g^\beta, g^{\alpha\beta}) = 1 : \alpha, \beta \xleftarrow{\$} \mathbb{Z}_q\right] \\ &\qquad - \Pr\left[\mathcal{D}(\mathbb{G}, q, g, g^\alpha, g^\beta, g^\gamma) = 1 : \alpha, \beta, \gamma \xleftarrow{\$} \mathbb{Z}_q\right] \Big| \\ \epsilon_{\text{CDH}}(\mathcal{D}) &:= \Pr\left[\mathcal{D}(\mathbb{G}, q, g, g^\alpha, g^\beta) = g^{\alpha\beta} : \alpha, \beta \xleftarrow{\$} \mathbb{Z}_q\right] \\ \epsilon_{\text{GDH}}(\mathcal{D}) &:= \Pr\left[\mathcal{D}^{\mathcal{O}_{\text{DDH}}}(\mathbb{G}, q, g, g^\alpha, g^\beta) = g^{\alpha\beta} : \alpha, \beta \xleftarrow{\$} \mathbb{Z}_q\right]\end{aligned}$$

We make use of the following cryptographic hardness assumptions:

(i) Decisional Diffie-Hellman (DDH): it is hard to distinguish $(g^\alpha, g^\beta, g^{\alpha\beta})$ from $(g^\alpha, g^\beta, g^\gamma)$, i.e., $\epsilon_{\text{DDH}}(\mathcal{D})$ is negligible in $\log(q)$ for any efficient $\mathcal{D}$.
(ii) Computational Diffie-Hellman (CDH): it is hard to compute the value $g^{\alpha\beta}$ from (g^α, g^β), i.e., $\epsilon_{\text{CDH}}(\mathcal{D})$ is negligible in $\log(q)$ for any efficient $\mathcal{D}$.

(iii) Gap Diffie-Hellman (GDH) [58]: it is hard to compute the value $g^{\alpha\beta}$ from (g^{α}, g^{β}) even when given black box access to a DDH oracle, i.e., $\epsilon_{\mathrm{GDH}}(\mathcal{D})$ is negligible in $\log(q)$ for any efficient $\mathcal{D}$.

We also make use of the random oracle model (ROM), instantiating all KDFs as black boxes which return a uniformly-random output for any given input.

Appendix B. Security Proof

The proof considers different cases corresponding to the possible behaviour of an adversary. We first describe the high-level proof structure in B.2. We then recall the main theorem and provide the actual proof in B.3.

B.1. Protocol Modifications for Key Indistinguishability

In order to apply a Bellare-Rogaway style key indistinguishability model for key exchange, in our proof we make two modifications to the Signal protocol. First, we remove all data messages from the protocol, considering only the key exchange messages. Second, we consider handshake messages as being sent in plain text instead of inside the associated data of the AEAD encryption of a message. Without this change, an adversary could distinguish a Tested message key from random by using it to verify the authentication of a handshake message.

B.2. Proof Structure Overview

Security in this sense means that no efficient adversary can break the multi-stage key-indistinguishability game for the two-party protocol Signal, parametrised by freshness condition fresh, with non-negligible probability. Suppose for contradiction that such an adversary $\mathcal{A}$ exists. Whatever the behaviour of the adversary, trivially (by the definition of the security experiment in Figure 6) it can only succeed when the Tested session $[s]$ is fresh. By Definition 4, this means that the $\mathsf{Test}(u, i, s)$ query satisfies:

(i) $\pi_u^i.\mathit{status}[s] = \mathtt{accept}$,
(ii) $\neg\pi_u^i.\mathsf{rev\_session}[s]$,
(iii) for all j such that $\pi_u^i.\mathit{sid}[s] = \pi_v^j.\mathit{sid}[s]$, $\neg\pi_v^j.\mathsf{rev\_session}[s]$, and
(iv) $\mathsf{clean}_{\pi_u^i.\mathit{type}[s]}(u, i, s)$

where v denotes $\pi_u^i.\mathsf{peerid}$, the identity of the intended peer to the Tested session, and $\mathsf{clean}_{\pi_u^i.\mathit{type}[s]}(u, i, s)$ is a cleanness clause as referenced in Definition 4 and subsequent definitions, further restricting the adversary's behaviour. In the following overview, we consider the case that the Tested session is the initiator; the responder is analogous.

Overview of the Case Distinction

A high-level overview of the proof with its main game sequences and case distinctions is given in Figure 7. Signal has many different types of stage, and we analyse each of them separately. Formally, we start with a sequence of generic game hops which apply to all cases, ruling out for instance the low-probability event that two randomly-generated DH keys collide. We then guess which session the adversary will choose as the Test session; since there can be only polynomially-many sessions this guess succeeds with non-negligible probability. (This is the source of the looseness in the proof.)

We then make a case distinction based on the stage type of the Tested session. Each stage type has its own cleanness predicate, and we deal with them in subcases. For example, if the adversary issues a query $\mathsf{Test}(u, i, [0])$, then we are in the analysis of stage $[0]$, and if the stage type is $\mathtt{triple}$, then we consider in turn the subcases where $\mathsf{clean}_{\mathtt{LM}}(u, i)$, $\mathsf{clean}_{\mathtt{EL}}(u, i, [0])$, or $\mathsf{clean}_{\mathtt{EM}}(u, i, [0])$ are true.

For each subcase, we perform an additional game hop, relying on one of the security assumptions in the statement of the theorem. The initial game will be ms-ind, and in the final games the session key will be replaced by a uniformly random value. By summing up the advantages along the way, we can obtain an overall bound on the success probability of the adversary.

The game hops in each of the subcases all build on a core type of reduction to GDH: the simulator queries for challenge values from the GDH oracle, inserting them into the game in place of certain DH keys and simulating the responses. We then intercept all adversarial calls to the random oracle, extracting the value taking the place of the GDH solution, and apply the DDH oracle to decide whether this value is indeed a solution. If it is then the simulator has broken GDH; if not, we continue the simulation. We will show that replacing the keys is not detectable by the adversary, and that violations of key indistinguishability imply solutions of the GDH challenge.

The challenger does not know in advance which adversary behaviour it is up against—only at the end of the game will the challenger know which of the clean predicates were satisfied. That is, the adversary might

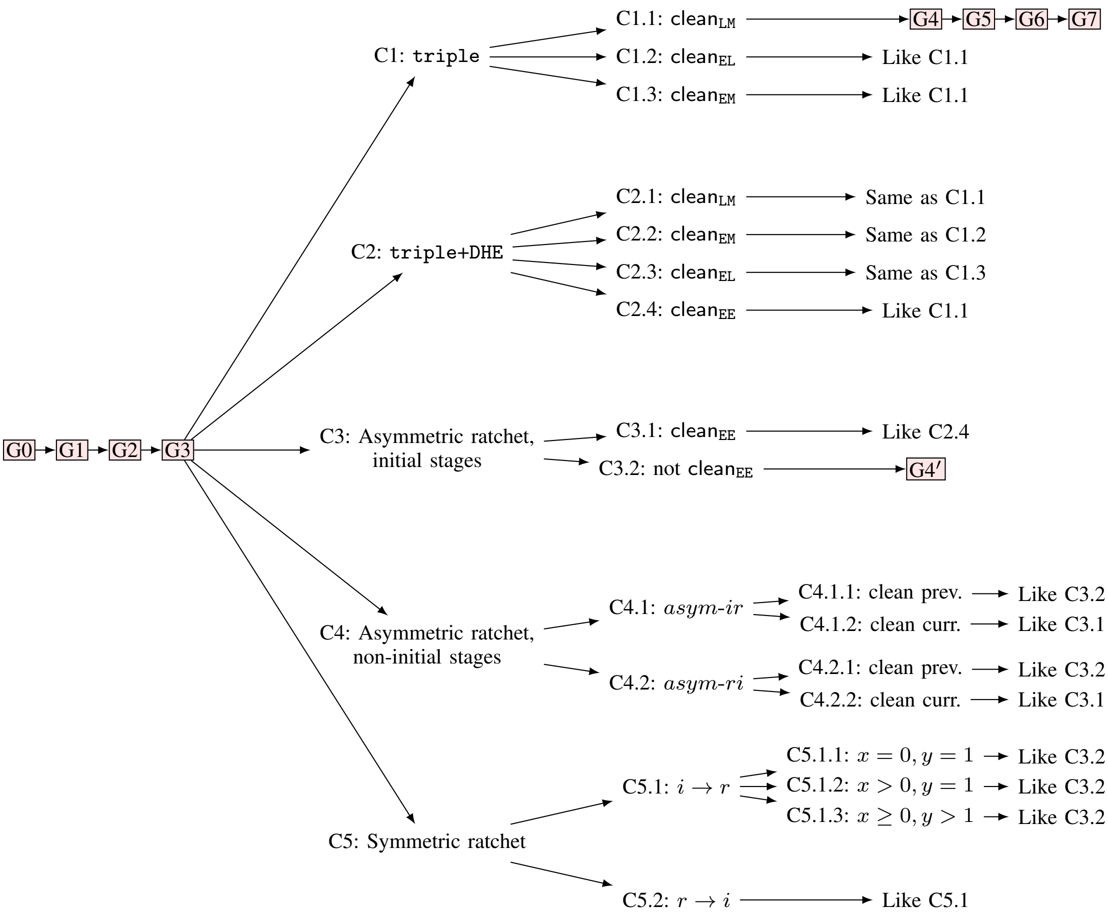


Figure 7: High-level overview of the proof structure. Games are identified by G0, G1, ..., and main case distinctions by C1, C2, ...; G0 denotes the multi-stage security experiment from Section 4.2. In the PDF version of this document, such identifiers can be clicked to jump to the corresponding part of the proof.

decide on the fly which session to Test and which clean predicate to satisfy. As such, no global reduction can be given. This is not a problem: in our proof, we are ultimately just ruling out classes of attacks. If an attack exists, it corresponds to some specific adversary, which is covered by one of our cases.

Cases. We begin by considering the case that the $\mathsf{Test}(u, i, s)$ query was issued on the first stage ($s = [0]$). We break this up into two separate cases:

(i) the initial key exchange had stage type `triple` (so 3 separate pairs of DH shared secrets were used to compute the master secret ms), or

(ii) the initial key exchange had stage type `triple+DHE` (so 4 separate pairs of DH shared secrets were used to compute the master secret ms).

Case 1: `triple`. In the first case, where $\mathsf{Test}(u, i, [0])$ and $\pi_u^i.\mathit{type}[0] = \mathtt{triple}$, we see by Definition 5 that the following condition must be satisfied:

$$\mathsf{clean}_{\mathtt{LM}}(u, i) \qquad \vee \qquad \mathsf{clean}_{\mathtt{EL}}(u, i, [0]) \qquad \vee \qquad \mathsf{clean}_{\mathtt{EM}}(u, i, [0])$$

Case 2: `triple+DHE`. In the second case, where $\mathsf{Test}(u, i, [0])$ and $\pi_u^i.\mathit{type}[0] = \mathtt{triple+DHE}$, we see by Definition 6 that there is an additional disjunct $\mathsf{clean}_{\mathtt{EE}}(u, i, [0])$, and we must have

$$\mathsf{clean}_{\mathtt{LM}}(u, i) \qquad \vee \qquad \mathsf{clean}_{\mathtt{EL}}(u, i, [0]) \qquad \vee \qquad \mathsf{clean}_{\mathtt{EM}}(u, i, [0]) \qquad \vee \qquad \mathsf{clean}_{\mathtt{EE}}(u, i, [0])$$

We consider each of these cases in turn, and by a game-hopping argument replace the relevant keys by random values, allowing us subsequently to replace the session keys with random values.

Case 3: Asymmetric ratchet, initial stage. Next, we consider the security of the case that the $\mathsf{Test}(u, i, s)$ query was issued on the initial responder-to-initiator asymmetric stage $s = [\textbf{asym-ri}{:}1]$. We partition this into two cases corresponding to Definition 8: either the adversary has not issued queries that would break the cleanness of the root key from the first stage $s = [0]$; or the adversary did not inject malicious DH shares in either of the ephemeral shares used in the stage (which in particular, were generated in stage $s = [0]$). That is, we consider the case that $\mathsf{Test}(u, i, s = [\mathbf{asym\text{-}ri}{:}1])$ where $\pi_u^i.\mathit{type}[s] = \texttt{asym-ri}$, and apply Definition 8 to conclude that

$$\Big(\mathsf{clean}_{\pi_u^i.\mathit{type}[0]}(u, i, [0]) \quad \wedge \quad \mathsf{clean}_{\mathsf{state}}(u, i, [\mathbf{asym\text{-}ir}{:}1])\Big) \qquad \vee \qquad \mathsf{clean}_{\texttt{EE}}(u, i, 0, 0)$$

In a similar fashion to the argument for the initial handshake, we replace certain DH values by values from a GDH challenger, reducing indistinguishability of the session key of this stage to hardness of GDH.

Case 4: Asymmetric ratchet, non-initial stages. We continue, considering the security of the case that the $\mathsf{Test}(u, i, s)$ query was issued in the x^{th} asymmetric responder-to-initiator stage $s = [\mathbf{asym\text{-}ri}{:}x]$. That is, we consider the case that $\mathsf{Test}(u, i, s = [\mathbf{asym\text{-}ri}{:}x])$, where $x \geq 2$ and $\pi_u^i.\mathit{type}[s] = \texttt{asym-ri}$. By Definition 8, we conclude:

$$\Big(\mathsf{clean}_{\texttt{asym-ri}}(u, i, [\mathbf{asym\text{-}ri}{:}x-1]) \wedge \mathsf{clean}_{\mathsf{state}}(u, i, [\mathbf{asym\text{-}ir}{:}x])\Big) \vee \mathsf{clean}_{\texttt{EE}}(u, i, x-1, x-1)$$

and a similar argument holds.

We must also consider the case that the Test query was issued against an asymmetric stage of type `asym-ir` i.e. a stage used to derive keys for the initiator to encrypt for the responder. The argument in this case is analogous but the cleanness predicates are subtly different and again vary depending on whether the stage is the first of its type. However, the core argument remains the same: we replace certain keys in the Tested session with values from the GDH challenger, in such a way that distinguishing session keys from random would give a GDH advantage.

Case 5: Symmetric ratchet. Finally, we consider symmetric stages. We partition into two cases:

(i) the first symmetric stage $y = 1$, where security follows from the asymmetric stage before it (which could be either the initial handshake or an asymmetric stage)

(ii) a later symmetric stage $y > 1$, where security follows from cleanness of the previous symmetric update

B.3. Proof of the Main Theorem

Conventions. We remark on a few conventions which we adopt during the proof.

Many cases technically differ based on whether the actor of the Test session has the initiator or responder role. For example, the first session key derived by the initiator is from a sending chain, while the first one derived by the responder is from a receiving chain. Where the security arguments are identical except for obvious symmetries, we just consider the case of the initiator and leave the responder as analogous.

Signal uses HMAC and HKDF within the KDF invocations. We assume that the KDF invocations themselves (as defined in Figure 2) are random oracles, and thus need not make any assumptions on HMAC and HKDF specifically.

By $\Pr(break_i)$ we mean the probability that the adversary wins game G_i. We aim to show that $\Pr(break_0)$ is close to $1/2$. To avoid overfilling our subscripts, we overload where it is obvious which game is meant.

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.*

Proof. We begin by performing a series of game hops that affect all potential cases. After these, the game hops diverge depending on which case we are considering.

Game Hops for all Cases

Game 0. This game equals the multi-stage security experiment described in Section 4.2. As such the probability of the adversary winning this game is bounded above by $\Pr(break_0)$.

Game 1. In this game we ensure no collision of honestly generated DH public keys. Specifically, the challenger $\mathcal{C}$ maintains a list L of all DH private values (for *ik*, *prek*, *ek*, *eprek*, *rchk*) honestly generated during the game. If a DH private value appears twice, $\mathcal{C}$ aborts the simulation and the adversary automatically loses. For an adversary's execution during the game, let $\mathsf{n_P}$ denote the total number of parties, $\mathsf{n_S}$ the maximum number of sessions, $\mathsf{n_M}$ the maximum number of medium-term keys per party, and $\mathsf{n_s}$ the maximum number of stages. We note that there are $\mathsf{n_P}$ long-term keys in the game, a maximum of $\mathsf{n_M}$ medium-term keys generated for each of the $\mathsf{n_P}$ parties for a maximum of $\mathsf{n_M}\mathsf{n_P}$ medium-term keys, and a maximum of $\mathsf{n_s}$ ephemeral/ratchet keys per session for a total maximum of $\mathsf{n_S}\mathsf{n_s}$ ephemeral/ratchet keys. This means a total maximum of $\mathsf{n_P} + \mathsf{n_P}\mathsf{n_M} + \mathsf{n_S}\mathsf{n_s}$ DH keys in the list L, every pair of which must not collide. There are $\binom{|L|}{2}$ such pairs of DH keys to consider in the game. Each DH key in L is in the same group of order q so collides with another key in L with probability $1/q$. Therefore we have the following bound:

$$\Pr(break_0) \leq \frac{\binom{\mathsf{n_P}+\mathsf{n_P}\mathsf{n_M}+\mathsf{n_S}\mathsf{n_s}}{2}}{q} + \Pr(break_1)$$

We now know that from this game onwards each honestly generated DH public key is unique. In future game hops we will replace certain DH values with ones sampled by a GDH challenger; this means that if these replacement values collide, we must abort the game and will therefore be unable to answer the GDH challenge. This will appear in game $G4$. Luckily, the probability of the GDH challenger producing colliding GDH challenge values is negligible (probability $1/q$), as we will see.

Game 2. In this game, the challenger guesses in advance the session π_u^i against which the $\mathsf{Test}(u, i, s)$ query is issued: the challenger guesses a pair of indices $(u^*, i^*) \in [1..\mathsf{n_P}] \times [1..\mathsf{n_S}]$, and aborts (and the adversary automatically loses) if the adversary issues a Test query $\mathsf{Test}(u, i, s)$ where $(u, i) \neq (u^*, i^*)$. This will occur with probability $1/\mathsf{n_S}\mathsf{n_P}$, and hence:

$$\Pr(break_1) \leq \mathsf{n_S}\mathsf{n_P} \cdot \Pr(break_2)$$

We remark that the bound we prove in this hop is not tight, and refer the reader to [3] for further discussions and impossibility results regarding tightness.

Game 3. In this game, the challenger guesses an index $v^* \in [\mathsf{n_P}]$ and aborts if there exists a session π_v^j that matches the Test session π_u^i but $v \neq v^*$. Note that it might be the case that no such matching π_v^j exists, but this game ensures that if such a π_v^j does exist, v is unique and known in advance by the challenger.

We must first show that there can exist at most one identity v with the same session identifier as π_u^i (note v may have multiple sessions that match π_u^i as the responder does not contribute freshness in the Triple-DH case). Alice's session identifier for stage $[0]$ contains ipk_v (the identity public key of the peer). In $G1$ we ensured that all DH values were unique, and hence the claim holds.

It follows that the challenger's guess is correct with probability $1/\mathsf{n_P}$, and so:

$$\Pr(break_2) \leq \mathsf{n_P} \cdot \Pr(break_3)$$

In this game, we do not guess the partner session because the responder does not always contribute an ephemeral key. As such, it is perfectly possible for v to have multiple sessions that match the test π_u^i because the adversary may replay π_u^i's ephemeral key to multiple sessions of v, which only uses the same public key and medium term key. Only in `triple+DHE` does v contribute a ephemeral key (that is unique due to Game 1) and indeed in this case we will do another game hop to guess the unique partner session in advance.

Currently, we have derived the following probability bound:

$$\Pr(break_0) \leq \frac{\binom{\mathsf{n_P}+\mathsf{n_P}\mathsf{n_M}+\mathsf{n_S}\mathsf{n_s}}{2}}{q} + \mathsf{n_S}\mathsf{n_P}^2 \cdot \Pr(break_3)$$

At this point, we need to partition our analysis for individual cases, with the ultimate aim of bounding $\Pr(break_3)$ above, which is upper bounded by the maximum success probability of the adversary in each case. Once we have bounded $\Pr(break_3)$, then we have bounded $\Pr(break_0)$ and we are done. Since this is $G3$, each different case begins with a hop to some $G4$.

C1: Initial key exchange: *type*$[0] =$ `triple`

First, we consider the security of Signal in the multi-stage key-indistinguishability game against an adversary $\mathcal{A}$ that issues a $\mathsf{Test}(u, i, [0])$ query with $\pi_u^i.type[0] = \mathtt{triple}$. By construction, the only way for the adversary to win (with non-negligible probability) is if $\mathsf{clean}_{\mathtt{triple}}(u, i, [0])$ is true. We partition these scenarios into subcases. Note also that a $\mathsf{RevState}(u, i, [0])$ or $\mathsf{RevState}(v, j, [0])$ (where π_v^j is a session matching π_u^i if one exists) query will reveal nothing to the adversary, as there exists no previous state. Moreover, after our game hops, we will have replaced the Tested message key with a uniformly random value that is independent to all other keys, so other issued RevState queries will only reveal independent root keys and chain keys. As the state will be independent from the Tested session key, it will not help the adversary distinguish the Test session key from random. How to simulate reveal queries will be dealt with formally in the game hops.

We now begin to separate our analysis based on sub-clauses of the cleanness predicate. Let $E^{\mathtt{triple}}$ be the event that an adversary $\mathcal{A}$ wins the ms-ind game by issuing a Test query $\mathsf{Test}(u, i, [0])$, such that

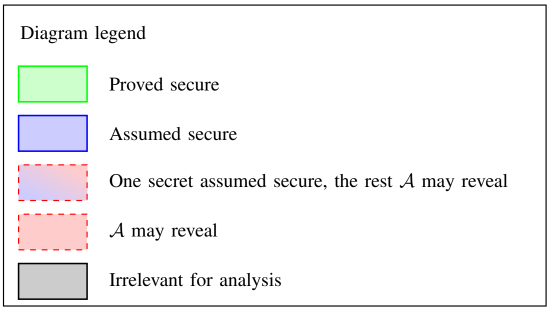


Figure 8: Legend for the boxes in the following diagrams. Red boxes indicate secrets that the adversary may gain access to via Reveal queries (or by computing the secrets as a result of the Reveal query), green boxes indicate secrets that are replaced based on the challenge, and blue boxes indicate secrets that the challenger is able to replace with random, thus ensuring security.

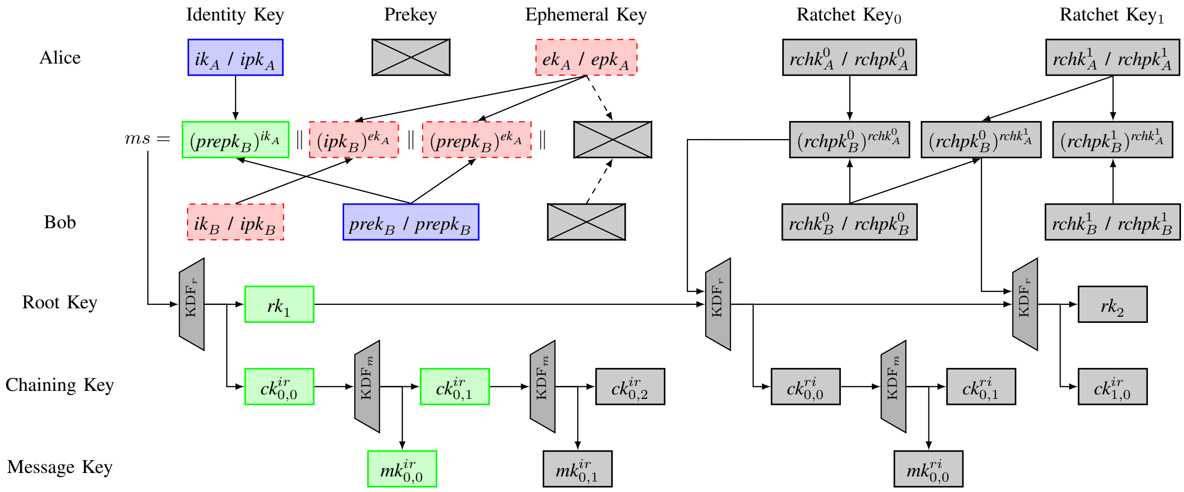


Figure 9: A diagram showing the replacement of secrets in Game 6 of Case 1.1. In particular, [red dashed box] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [green box] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [blue box] denotes secrets that the challenger is able to replace with random, thus ensuring security; [grey box] denotes secrets that are not relevant to this case; and [red/blue dashed box] denotes secrets one of which is assumed uncompromised but the rest may be revealed by $\mathcal{A}$.

$\pi_u^i.type = \texttt{triple}$, and let $E^{\texttt{triple}}_{\mathsf{clean}_{\mathtt{LM}}}$ (resp. $E^{\texttt{triple}}_{\mathsf{clean}_{\mathtt{EL}}}$, $E^{\texttt{triple}}_{\mathsf{clean}_{\mathtt{EM}}}$) be the sub-case in which additionally $\mathsf{clean}_{\mathtt{LM}}(u,i)$ (resp. $\mathsf{clean}_{\mathtt{EL}}(u,i,[0])$, $\mathsf{clean}_{\mathtt{EM}}(u,i,[0])$) is true. By definition of $\mathsf{clean}_{\texttt{triple}}$,

$$\Pr(E^{\texttt{triple}}) = \max\left\{\Pr\left(E^{\texttt{triple}}_{\mathsf{clean}_{\mathtt{LM}}(u,i)}\right), \Pr\left(E^{\texttt{triple}}_{\mathsf{clean}_{\mathtt{EL}}(u,i,0)}\right), \Pr\left(E^{\texttt{triple}}_{\mathsf{clean}_{\mathtt{EM}}(u,i,0)}\right)\right\}$$

C1.1: Case *type*$[0] = \texttt{triple}$ and $\mathsf{clean}_{\mathtt{LM}}(u,i)$:

In this case $\mathcal{A}_{\texttt{triple}}$ issued a $\mathsf{Test}(u,i,[0])$ query such that $\mathsf{clean}_{\mathtt{LM}}(u,i)$ is upheld. For Test sessions where $\pi_u^i.role = \texttt{init}$, this requires that $\mathcal{A}_{\texttt{triple}}$ has not issued $\mathsf{RevLongTermKey}(u)$ or $\mathsf{RevMedTermKey}(v,n)$ where $\pi_u^i.\mathsf{peerpreid} = n$. Since we do not consider the signatures over the medium-term prekeys in our model, we may assume that π_u^i has received $prepk_v^n$ without modification.

Recall that an honest session derives a master secret $ms = g^{ik_u \cdot prek_v^n} \| g^{ek_u \cdot ik_v} \| g^{ek_u \cdot prek_v^n}$, and then assigns $rk^1 \parallel ck^{\mathbf{sym\text{-}ir}:0,0} \leftarrow \mathrm{HKDF}(ms)$.

Our goal will be to replace the session key with a random value so that the adversary cannot guess the hidden bit (game 7). Since we are working in the random oracle model and the session key is the output of a call to the random oracle, this means the adversary must query the random oracle on the exact input (game 6). We embed a Gap Diffie–Hellman challenge into one of the components of the input to the random oracle. For this particular case (C1.1), which depends on the $\mathsf{clean}_{\mathtt{LM}}(u,i)$ condition, we embed the GDH challenge into the long-term key of party u and one of the medium-term keys of the peer v (hop from game 5 to game 6). In order do this embedding, we must guess which of the medium-term keys of the peer is actually used (game 4). (There is also a minor technicality covered in game 5, described below.)

Game 4. In this game, the challenger guesses the index $n \in [1..\mathsf{n_M}]$ of the signed prekey of the peer ($prek_v^n$) that the Test session will use in the execution of the protocol, and aborts if the guess is wrong. This yields that

$$\Pr(break_3) \le \mathsf{n_M} \cdot \Pr(break_4) \ .$$

Game 5. In this game, the experiment *does not* abort if ipk_u and $prepk_v^n$ are the same. (Recall that in Game 1, we added an abort event if any DH values were the same. We will soon want to employ a GDH challenger, but the two challenge public keys in a GDH challenger may (with small probability) be the same, so we need to re-allow that in our game hops.) Since the keys are elements of a group of order q, the probability that one of them equals the other is $1/q$ and thus

$$\Pr(break_4) \le 1/q + \Pr(break_5) \ .$$

Game 6. In this game the experiment aborts if the adversary queries $g^{ik_u \cdot prek_v^n} = \mathrm{CDH}(ipk_u, prepk_v^n)$ as the first component of a call to the HKDF random oracle; denote this event $abort_6$. Thus,

$$\Pr(break_5) \le \Pr(break_6) + \Pr(abort_6) \ .$$

We now need to bound $\Pr(abort_6)$. To do so, we show that, whenever event $abort_6$ occurs, we can construct an algorithm $\mathcal{B}_0$ that can win the Gap Diffie–Hellman problem. In the GDH experiment, $\mathcal{B}_0$ receives as input a DH pair (g^α, g^β) (for α and β unknown to $\mathcal{B}_0$), and has access to an oracle $\mathcal{O}_{\mathrm{DDH}}$ that on input (g^x, g^y, g^z) returns 1 if and only if $g^{xy} = g^z$.

$\mathcal{B}_0$ will simulate game 5, except that it replaces ipk_u with g^α and $prepk_v^n$ with g^β. Because certain keys have been replaced with public keys whose corresponding private values are unknown to $\mathcal{B}_0$, we must define the actions that should be taken when these private values would normally be used in a computation. Cleanness implies that $\neg\mathsf{rev\_ltk}_u \wedge \neg\mathsf{rev\_mtk}_v^n$, so $\mathcal{B}_0$ will not need to answer any Reveal queries from $\mathcal{A}$ on these values. However, since $\mathcal{B}_0$ has replaced the long-term identity key and a medium-term public key of two parties, if $\mathcal{A}$ decides to direct parties u or v to execute the protocol in a non-Tested session, then $\mathcal{B}_0$ may need to perform simulations of concrete computations with the private keys α and β, despite not knowing them. There are three distinct types of sessions in which $\mathcal{B}_0$ may lack the private keys needed to compute the master secret ms of that session:

(i) a non-Tested session between user u and user v using $prek_v^n$ where u is the initiator;
(ii) a non-Tested session between user u and some other user (possibly v or not) where u is the responder;
(iii) a session between a user other than u and user v using $prek_v^n$ where v is the responder.

In session type (i), the simulator does not know $\mathrm{CDH}(g^\alpha, g^\beta)$ which would be an input to the KDF computation of the session key (in fact this is the value that the simulator needs to find in the GDH game). In session type (ii), the simulator does not know $\mathrm{CDH}(g^\alpha, g^e)$ for unknown, potentially maliciously chosen, e. In session type (iii), the simulator does not know $\mathrm{CDH}(g^\beta, g^e)$ for unknown, potentially maliciously chosen, e.

In each of these types of sessions, $\mathcal{B}_0$ will pick random keys $rk^1, ck^{\mathbf{sym\text{-}ir}:0,0}$ rather them deriving than via $\mathrm{HKDF}(ms)$. $\mathcal{B}_0$ maintains a list of all sessions in which random keys have been substituted: the list contains the random session keys as well as the public keys that should have been used to compute each component of the master secret. $\mathcal{B}_0$ must also ensure that key values used are consistent with any queries that $\mathcal{A}$ makes to the random oracle HKDF. We are concerned about queries of the form $g^{x_1} \| g^{x_2} \| g^{x_3}$. Before answering any such query, $\mathcal{B}_0$ goes through each entry in the above list of sessions: for each entry in the list, it uses its DDH oracle to check if the public keys that should have been used to compute each component of the master secret match the corresponding component (g^{x_1}, g^{x_2} or g^{x_3}) of this random oracle query. For example, for session type (i) this amounts to querying the DDH oracle $\mathcal{O}_{\mathrm{DDH}}(g^\alpha, g^\beta, g^{x_1})$ and possibly $\mathcal{O}_{\mathrm{DDH}}(g^e, g^\beta, g^{x_2})$. If all components, when queried in the DDH oracle, return 1, then $\mathcal{B}_0$ uses the randomly chosen keys from that element of the list as the random oracle response; otherwise, $\mathcal{B}_0$ samples a new random value as the random oracle response. Similarly session types (ii) and (iii) can be simulated.

(While the explanation above starts from $\mathcal{B}_0$ picking random session keys when simulating a session and then ensuring random oracle queries are answered consistently, $\mathcal{B}_0$ must also do the reverse: when simulating a session, before picking random keys $\mathcal{B}_0$ analogously use its DDH oracle to check if this matches a previous random oracle query, to ensure correct simulation.)

Note the session type (i) is special: if $\mathcal{O}_{\mathrm{DDH}}(g^\alpha, g^\beta, g^{x_1}) = 1$, then the adversary has found the solution to the GDH problem for us, and $\mathcal{B}_0$ can use g^{x_1} as its answer to the GDH challenger. Moreover, this is exactly when the event $abort_6$ occurs.

$$\Pr(abort_6) = \epsilon_{\mathrm{GDH}}(\mathcal{B}_0) \ .$$

Game 7. In this game, the experiment replaces the session key in the Test session with a uniformly random key from the same space. Because of the abort in game 6, we know that the adversary never queried the random

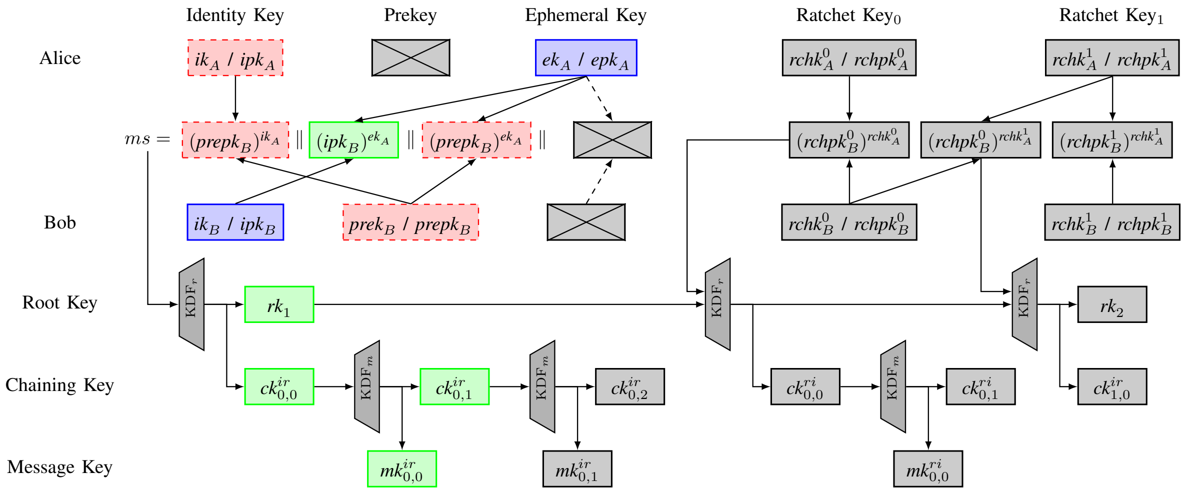


Figure 10: A diagram showing the replacement of secrets in Game 5 of Case 1.2. In particular, [red dashed box] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [green box] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [blue box] denotes secrets that the challenger is able to replace with random, thus ensuring security; [grey box] denotes secrets that are not relevant to this case; and [dashed box] denotes secrets one of which is assumed uncompromised but the rest may be revealed by $\mathcal{A}$.

oracle HKDF on the input ms that was used to compute the session key $rk^1 \parallel ck^{\mathbf{sym\text{-}ir}:0,0}$ of the Test session. Thus, in the random oracle model,

$$\Pr(break_6) = \Pr(break_7) \ .$$

Finally, since the session key is uniformly random and independent of the hidden bit, the adversary has no advantage in guessing the hidden bit and winning the experiment:

$$\Pr(break_7) = 1/2 \ .$$

C1.2: Case $\textit{type}[0] = \texttt{triple}$ and $\mathsf{clean}_{\mathtt{EL}}(u, i, 0)$

In this case the adversary $\mathcal{A}_{\mathtt{triple}}$ has issued a Test query $\mathsf{Test}(u, i, [0])$ such that $\mathsf{clean}_{\mathtt{EL}}(u, i, [0])$ is upheld. For Test sessions such that $\pi_u^i.\textit{role} = \texttt{init}$, this means that $\mathcal{A}_{\mathtt{triple}}$ has not issued $\mathsf{RevRand}(u, i, [0])$ and $\mathsf{RevLongTermKey}(v)$ where $v = \pi_u^i.\textit{peeripk}$. For Test sessions such that $\pi_u^i.\textit{role} = \texttt{resp}$, this means that $\mathcal{A}_{\mathtt{triple}}$ has not issued $\mathsf{RevLongTermKey}(u)$ and a $\mathsf{RevRand}(v, j, [0])$ such that $\pi_v^j.\textit{sid}[0]$ matches the Test session $\pi_u^i.\textit{sid}[0]$.

Game 4, 5, 6. The argument for this case is almost identical to that of the previous case, except we no longer need to guess the index of the long-term key of the responder or the ephemeral key of the initiator. The GDH challenge values g^α, g^β are inserted into the simulation in Game 5 in place of the ephemeral key of the initiator and the long-term key of the responder. Thus

$$\Pr(break_3) \leq 1/q + \epsilon_{\mathrm{GDH}} + 1/2$$

C1.3: Case $\textit{type}[0] = \texttt{triple}$ and $\mathsf{clean}_{\mathtt{EM}}(u, i, [0])$

Game 4, 5, 6, 7. In this case, the adversary $\mathcal{A}_{\mathtt{triple}}$ has issued a Test query $\mathsf{Test}(u, i, [0])$ such that $\mathsf{clean}_{\mathtt{EM}}(u, i, [0])$ is upheld. For Test sessions such that $\pi_u^i.\textit{role} = \texttt{init}$, this means that $\mathcal{A}_{\mathtt{triple}}$ has not issued a $\mathsf{RevRand}(u, i, 0)$ and $\mathsf{RevMedTermKey}(v, \pi_u^i.\mathsf{peerpreid})$. For Test sessions such that $\pi_u^i.\textit{role} = \texttt{resp}$, this means that $\mathcal{A}_{\mathtt{triple}}$ has not issued a $\mathsf{RevRand}(v, j, [0])$ such that $\pi_v^j.\textit{sid}[0]$ matches the Test session $\pi_u^i.\textit{sid}[0]$ and $\mathsf{RevMedTermKey}(u, \pi_u^i.\textit{prepk})$.

Again, this is analogous to before. We begin by guessing the index of the signed prekey of the responder, incurring a factor of $\mathsf{n_M}$. The Gap-DH challenge values g^α, g^β are inserted into the simulation in Game 6 in place of the ephemeral key of the initiator and the particular medium-term key of the responder used in the Test session. Thus

$$\Pr(break_3) \leq \mathsf{n_M} \cdot (1/q + \epsilon_{\mathrm{GDH}}) + 1/2$$

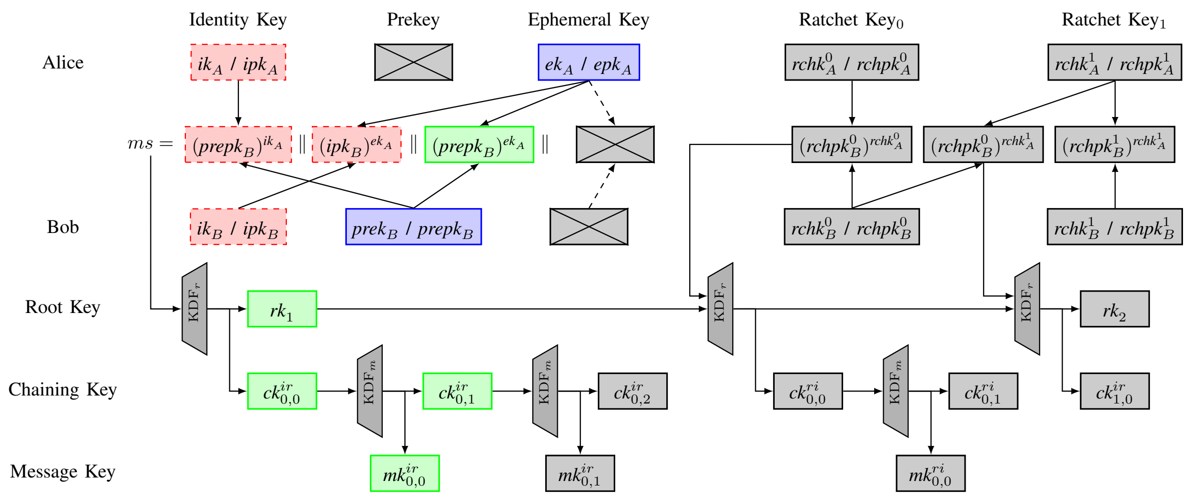


Figure 11: A diagram showing the replacement of secrets in Game 6 of Case 1.3. In particular, [red dashed box] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [green box] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [blue box] denotes secrets that the challenger is able to replace with random, thus ensuring security; [grey box] denotes secrets that are not relevant to this case; and [purple dashed box] denotes secrets one of which is assumed uncompromised but the rest may be revealed by $\mathcal{A}$.

C2: Initial key exchange: *type*$[0] = \mathtt{triple+DHE}$

Recall that the initial key exchange can also have type $\mathtt{triple+DHE}$, in which case cleanness requires that

$$\mathsf{clean}_{\mathtt{LM}}(u,i) \vee \mathsf{clean}_{\mathtt{EL}}(u,i,[0]) \vee \mathsf{clean}_{\mathtt{EM}}(u,i,[0]) \vee \mathsf{clean}_{\mathtt{EE}}(u,i,[0])$$

We now consider the case that the adversary has issued a Test query $\mathsf{Test}(u,i,[0])$ where the stage $\pi^i_u.\mathit{type}[0] = \mathtt{triple+DHE}$. We note that the cases are the same as previously, with the additional case $\mathsf{clean}_{\mathtt{EE}}(u,i,[0])$. As before, we define

- $E^{\mathtt{triple+DHE}}_{\mathsf{clean}_{\mathtt{LM}}}$ to be the event that an adversary wins the multi-stage key-indistinguishability game where $\mathcal{A}$ has issued a Test query $\mathsf{Test}(u,i,[0])$ and $\mathsf{clean}_{\mathtt{LM}}(u,i)$ is upheld,
- $E^{\mathtt{triple+DHE}}_{\mathsf{clean}_{\mathtt{EM}}}$ where $\mathcal{A}$ has issued a Test query $\mathsf{Test}(u,i,[0])$ and $\mathsf{clean}_{\mathtt{EM}}(u,i,[0])$ is upheld,
- $E^{\mathtt{triple+DHE}}_{\mathsf{clean}_{\mathtt{EL}}}$ where $\mathcal{A}$ has issued a Test query $\mathsf{Test}(u,i,[0])$ and $\mathsf{clean}_{\mathtt{EL}}(u,i,[0])$ is upheld, and
- $E^{\mathtt{triple+DHE}}_{\mathsf{clean}_{\mathtt{EE}}}$ where $\mathcal{A}$ has issued a Test query $\mathsf{Test}(u,i,[0])$ and $\mathsf{clean}_{\mathtt{EE}}(u,i,[0],[0])$ is upheld.

We will bound the adversary's success probability as follows.

$$\Pr(E^{\mathtt{triple+DHE}}) = \max\left\{\Pr(E^{\mathtt{triple+DHE}}_{\mathsf{clean}_{\mathtt{LM}}(u,i)}), \Pr(E^{\mathtt{triple+DHE}}_{\mathsf{clean}_{\mathtt{EL}}(u,i,[0])}), \Pr(E^{\mathtt{triple+DHE}}_{\mathsf{clean}_{\mathtt{EM}}(u,i,[0])}), \Pr(E^{\mathtt{triple+DHE}}_{\mathsf{clean}_{\mathtt{EE}}(u,i,[0],[0])})\right\}$$

The bounds for the previous summands are proved to be negligible under our cryptographic assumptions exactly as above, yielding the inequalities as desired. As before, the crucial proof step in each case is the Gap-DH assumption. However, for this case it will also make a game hop like Game 3, where we additionally know Bob's unique matching session in advance. We can do this now because Bob has freshness in the handshake.

C2.4: Case *type*$[0] = \mathtt{triple+DHE}$ and $\mathsf{clean}_{\mathtt{EE}}$

Game 4, 5, 6, 7. The final ephemeral-ephemeral case $E^{\mathtt{triple+DHE}}_{\mathsf{clean}_{\mathtt{EE}}}$ is analogous to previous cases except that in Game $G6$ ($E^{\mathtt{triple+DHE}}_{\mathsf{clean}_{\mathtt{EE}}}$), we need to replace the ephemeral values of both the initiator and the responder. (Since the simulator in $G4$ will never reuse ephemeral values in a different session, the simulation in this case is simpler and will not need to use its DDH oracle to maintain consistency.) We have to consider that the responder party generates a list of one-time ephemeral keys that new sessions (used in sessions executed by the responder) may use, and thus $G4$ now incurs a factor of $\mathsf{n_S}$. Thus

$$\Pr(break_3) \leq \mathsf{n_S} \cdot (1/q + \epsilon_{\mathrm{GDH}}) + 1/2$$

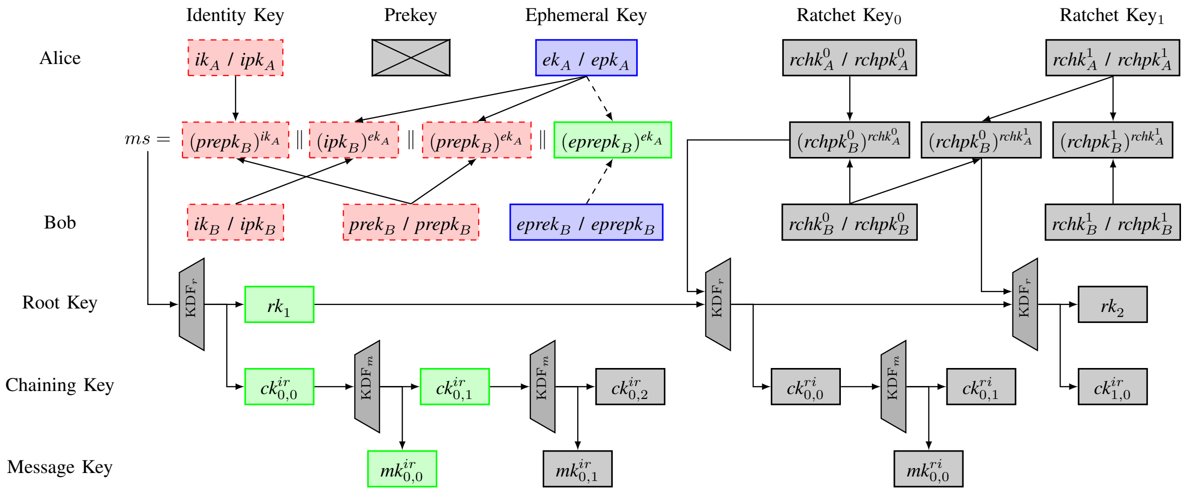


Figure 12: A diagram showing the replacement of secrets in Game 6 of Case 2.4. In particular, [red dashed box] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [green box] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [blue box] denotes secrets that the challenger is able to replace with random, thus ensuring security; [grey box] denotes secrets that are not relevant to this case; and [red/blue dashed box] denotes secrets one of which is assumed uncompromised but the rest may be revealed by $\mathcal{A}$.

C3: Asymmetric ratcheting, initial stage

We have now proved security of the initial key exchange, optionally including the ephemeral-ephemeral DH computation. We next move on to the asymmetric-ratcheting stages, in which Bob and Alice take turns generating new DH ephemerals and updating their root keys. The first asymmetric-ratcheting stage differs slightly from its successors since it immediately follows the initial handshake, and we deal with it here now. Recall it is of type `asym-ri`, since it is performed when Bob wishes to send a message to Alice.

We consider an adversary $\mathcal{A}$ that issues a $\mathsf{Test}(u, i, s = [\textbf{asym-ri}{:}1])$ query, where stage s must have *type* $=$ `asym-ri`. Note that the initial asymmetric stage is always of type `asym-ri` (messages from Alice to Bob before this stage are sent using the symmetric chain derived from the initial handshake), and thus in this section we do not need to consider *initial* stages of type `asym-ir`. We define

- $E^{\texttt{asym-ri}}$ to be the event that an adversary $\mathcal{A}$ wins the multi-stage key-indistinguishability game by issuing a $\mathsf{Test}(u, i, s = [\textbf{asym-ri}{:}1])$ query,
- $E^{\texttt{asym-ri}}_{\mathsf{clean}_{\mathtt{EE}}(u,i,[0])}$ to be the sub-event of $E^{\texttt{asym-ri}}$ satisfying $\mathsf{clean}_{\mathtt{EE}}(u, i, [0], [0])$, and
- $E^{\texttt{asym-ri}}_{\mathsf{clean}_{\pi_u^i.\mathit{type}[0]}(u,i,[0])}$ to be the sub-event of $E^{\texttt{asym-ri}}$ satisfying

$$\mathsf{clean}_{\pi_u^i.\mathit{type}[0]}(u, i, [0]) \wedge \mathsf{clean}_{\mathsf{state}}(u, i, [\textbf{asym-ri}{:}1]).$$

It follows from our definition of freshness that

$$\Pr(E^{\texttt{asym-ri}}) \leq \max\left\{\Pr(E^{\texttt{asym-ri}}_{\mathsf{clean}_{\mathtt{EE}}(u,i,[0])}), \Pr(E^{\texttt{asym-ri}}_{\mathsf{clean}_{\pi_u^i.\mathit{type}[0]}(u,i,[0])})\right\} \tag{1}$$

and we consider these two cases in turn, beginning with the case that $\mathsf{clean}_{\mathtt{EE}}(u, i, [0], [0])$ is upheld.

C3.1: Case $s = [\textbf{asym-ri}{:}1]$, *type*$[s] =$ `asym-ri` and $E^{\texttt{asym-ri}}_{\mathsf{clean}_{\mathtt{EE}}}$

Game 4, 5, 6, 7. This case is dealt with similarly to case 2.4, with the only substantial differences being that the GDH challenge values are substituted into the ratchet keys $g^{\mathit{rchk}^u_{[0]}}$ and $g^{\mathit{rchk}^v_{[0]}}$ and we do not need to guess the index of the ratchet keys. Since we have a randomness-reveal query instead of queries for specific keys, the predicate $\mathsf{clean}_{\mathtt{EE}}$ covers secrecy both of the handshake ephemerals and of the initial ratchet keys, which are generated at the same time. Thus

$$\Pr(\mathit{break}_3) \leq {}^1\!/\!_q + \epsilon_{\mathsf{GDH}} + {}^1\!/\!_2$$

C3.2: Case $s = [\textbf{asym-ri}{:}1]$, *type*$[s] =$ `asym-ri` and $E^{\texttt{asym-ri}}_{\mathsf{clean}_{\pi_u^i.\mathit{type}[0]}(u,i,[0])}$

In this case, cleanness comes from the initial key exchange (i.e., from one of its three or four disjuncts), and the fact that the adversary has not revealed the state linking the initial key exchange to this stage. The

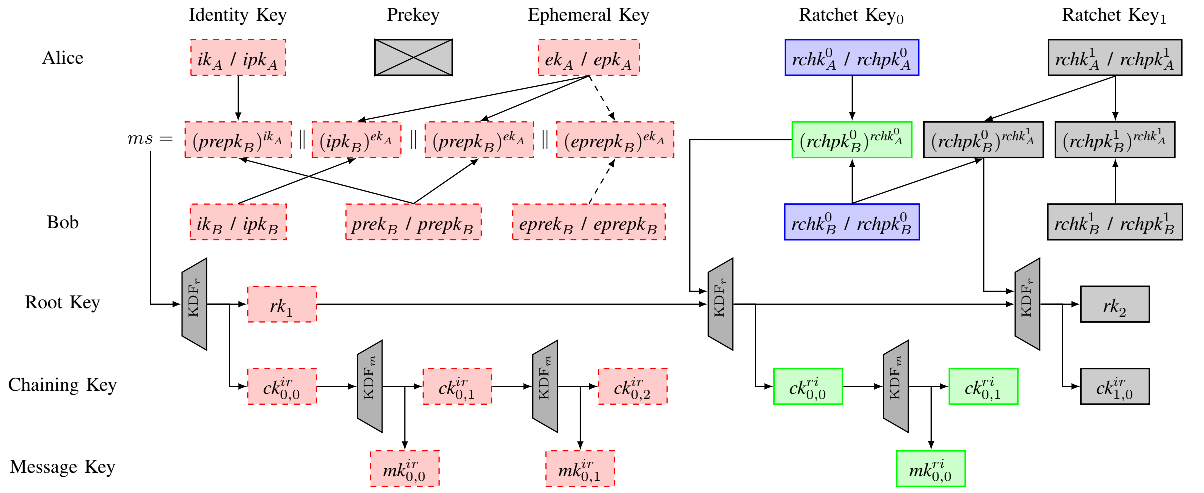


Figure 13: A diagram showing the replacement of secrets in Game 6 of Case 3.1. In particular, denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; denotes secrets that the challenger is able to replace with random, thus ensuring security; denotes secrets that are not relevant to this case; and denotes secrets one of which is assumed uncompromised but the rest may be revealed by $\mathcal{A}$.

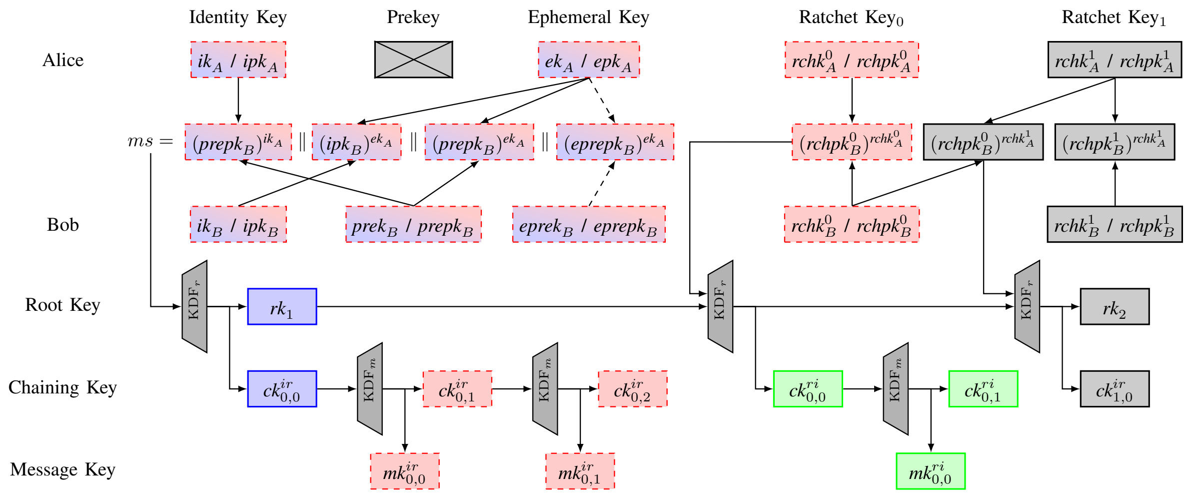


Figure 14: A diagram showing the replacement of secrets in Game 7' of Case 3.2. In particular, denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; denotes secrets that the challenger is able to replace with random, thus ensuring security; denotes secrets that are not relevant to this case; and denotes secrets one of which is assumed uncompromised but the rest may be revealed by $\mathcal{A}$.

initial key exchange derives rk_1: we perform one game hop to replace that value with a uniformly random value; the game hop is indistinguishable assuming the security of rk_1, which follows from Cases 1 and 2. Game $7'$ is indicated below.

Game $7'$**.** In this game we replace the root key rk_1 derived in stage $[0]$ by both the Test session and any matching peers with a uniformly random value. Thus $\Pr(break_{7'})$.

An adversary which can distinguish $G7'$ from its predecessor game can distinguish the root key from a random value. The root key was derived in the initial `triple` (or `triple+DHE`) handshake by applying KDF_r to the master secret ms. In Case 1 (or Case 2), we argued that all values derived from ms using HKDF were indistinguishable from random. Thus, an adversary that wins here contradicts the security of Case 1 (or Case 2). Recall that we denote with $\Pr(E^{\texttt{triple}})$ the adversary's probability of success in breaking the key-indistinguishability of Case 1, and with $\Pr(E^{\texttt{triple+DHE}})$ we denote the adversary's probability of success in breaking the key-indistinguishability of Case 2. Given that only one of Case 1 or Case 2 applies (given how the initial key exchange is either of type `triple` or type `triple+DHE`), then the adversary's probability in distinguishing this change is $\max\{\Pr(E^{\texttt{triple}}), \Pr(E^{\texttt{triple+DHE}})\}$. Note that $\Pr(E^{\texttt{triple}}) \leq \Pr(E^{\texttt{triple+DHE}})$, given that $\Pr(E^{\texttt{triple}}) = \max\{\Pr(E^{\texttt{triple}}_{\texttt{clean}_{\texttt{LM}}(u,i)}), \Pr(E^{\texttt{triple}}_{\texttt{clean}_{\texttt{EL}}(u,i,0)}), \Pr(E^{\texttt{triple}}_{\texttt{clean}_{\texttt{EM}}(u,i,0)})\}$ and $\Pr(E^{\texttt{triple+DHE}}) =$

$\max\{\Pr(E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\mathsf{LM}}(u,i)}), \Pr(E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\mathsf{EL}}(u,i,[0])}), \Pr(E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\mathsf{EM}}(u,i,[0])}), \Pr(E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\mathsf{EE}}(u,i,[0],[0])})\}$, and that the upper bounds on the first three subcases of both Case 1 and Case 2 are identical.

After replacing the root key rk_1, it is straightforward to see that it is impossible for the adversary to differentiate keys derived in this stage—chaining keys $ck^{\textbf{sym-ri}:x,0}$ and $ck^{\textbf{sym-ri}:x,1}$, messaging key $mk^{\textbf{sym-ri}:x,0}$, and intermediate value tmp from random: these are derived by applying $\mathrm{KDF_r}$ to rk_1 and then $\mathrm{KDF_m}$ to that result. Since both KDFs are modelled as random oracles, and the input to $\mathrm{KDF_r}$ is an independent uniformly random value, the adversary has no advantage is distinguishing this stage's session key from random. Thus

$$\Pr(break_{7'}) = \Pr(E^{\texttt{triple+DHE}}) + 1/2$$

C4: Asymmetric ratcheting, non-initial stages

At this point we move on to arbitrary subsequent asymmetric stages. We assume that the initial handshake was of type `triple`, but the case of `triple+DHE` is analogous. The intuition for this part of the proof is essentially induction and post-compromise security:

- root keys provide security because they come from previous stages which are secure; or
- shared secrets derived from pairs of ephemeral keys provide security even if the root key at the time is compromised.

We first make a case distinction on the direction (`asym-ir` vs. `asym-ri`), and then deal with these cases in turn.

C4.1: Asymmetric ratcheting, $s = [\textbf{asym-ir}{:}x], x \geq 1$, *type*$[s] = \texttt{asym-ir}$

Definition 8 requires that one of the following conditions must be satisfied if $\mathsf{clean}_{\texttt{asym-ir}}(u, i, [\textbf{asym-ir}{:}x])$ is to hold.

- event $E^{\texttt{asym-ir}}_{\text{clean-prev}}$: $\mathsf{clean}_{\texttt{asym-ri}}(u, i, [\textbf{asym-ri}{:}x-1]) \wedge \mathsf{clean}_{\mathsf{state}}(u, i, [\textbf{asym-ir}{:}x])$
- event $E^{\texttt{asym-ir}}_{\text{clean-cur}}$: $\mathsf{clean}_{\mathsf{EE}}(u, i, x-1, x-1)$

C4.1.1: Case $s = [\textbf{asym-ir}{:}x], x \geq 1$, *type*$[s] = \texttt{asym-ir}$ and $E^{\texttt{asym-ir}}_{\textbf{clean-prev}}$

This case follows inductively like Case 3.2. This stage's message key (as well as the next root and chaining key) is derived by applying $\mathrm{KDF_r}$ to tmp, which was derived during stage $[\textbf{asym-ri}{:}x]$, and then $\mathrm{KDF_m}$ to the result. By an argument similar to Case 3.2, we can replace tmp with a random key. Treating the KDF as a random oracle, this stage's message key, as well as the next root key rk_{x+1} and the symmetric chaining keys $ck^{\textbf{sym-ir}:x,0}$ and $ck^{\textbf{sym-ir}:x,1}$, are then indistinguishable from random. Thus

$$\Pr(break_3) = 1/q + \Pr(E^{\texttt{triple+DHE}}) + \epsilon_{\text{GDH}} + 1/2$$

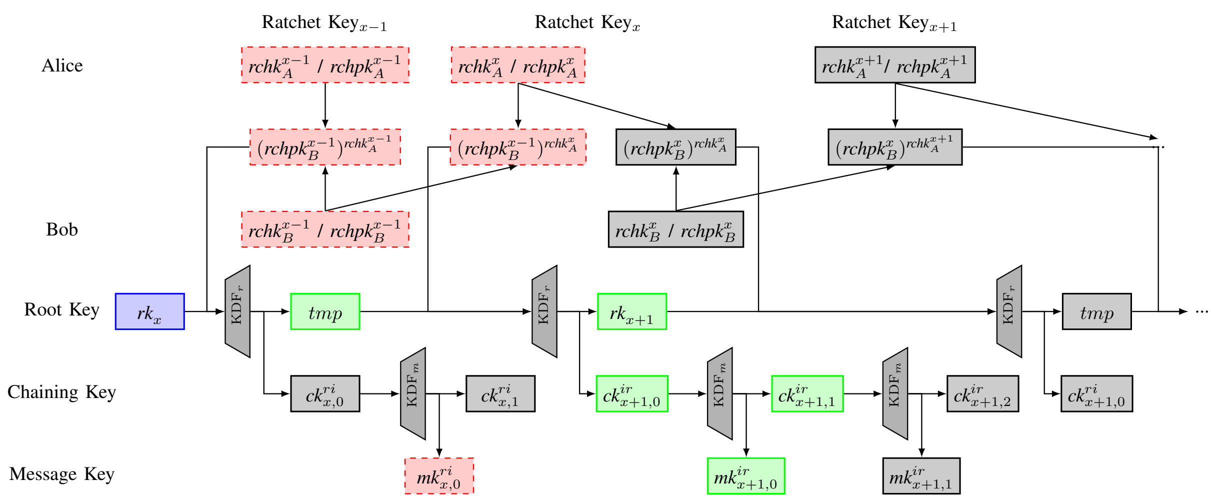


Figure 15: A diagram showing the replacement of secrets in Game 7' of Case 4.1.1. In particular, [red dashed box] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [green box] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [blue box] denotes secrets that the challenger is able to replace with random, thus ensuring security; [grey box] denotes secrets that are not relevant to this case; and [red/blue dashed box] denotes secrets one of which is assumed uncompromised but the rest may be revealed by $\mathcal{A}$.

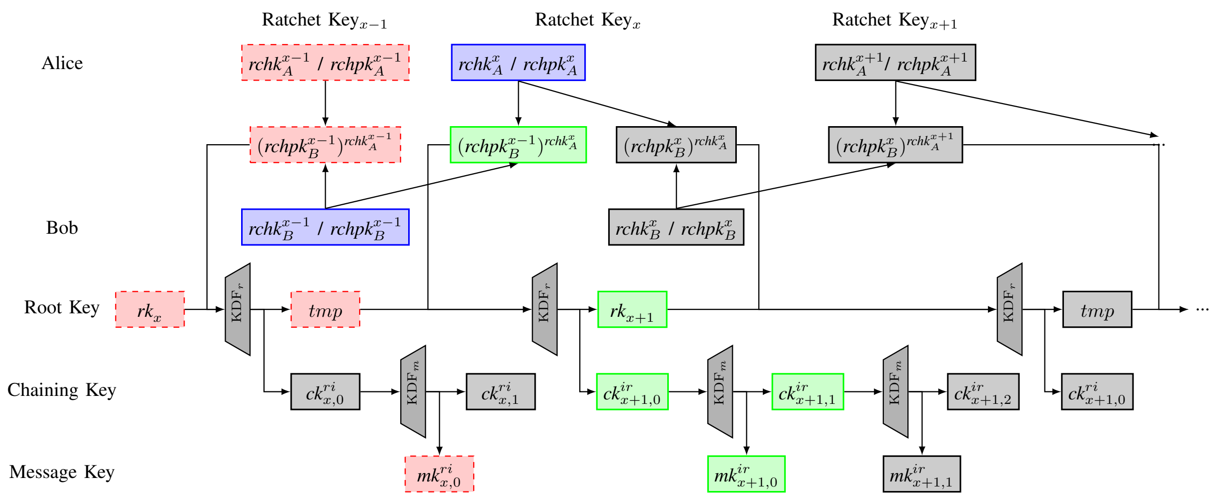


Figure 16: A diagram showing the replacement of secrets in Game 7' of Case 4.1.2. In particular, [ ] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [ ] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [ ] denotes secrets that the challenger is able to replace with random, thus ensuring security; [ ] denotes secrets that are not relevant to this case; and [ ] denotes secrets one of which is assumed uncompromised but the rest may be revealed by $\mathcal{A}$.

C4.1.2: Case $s = [\textbf{asym-ir:}x], x \geq 1$, *type*$[s] = \texttt{asym-ir}$ and $E^{\texttt{asym-ir}}_{\textbf{clean-cur}}$

This case is analogous to Case 3.1, with key indistinguishability following from secrecy of the DH shared secret derived from ratchet keys. We first replace the ratchet public keys with challenge values from the Gap-DH game, noting that $\mathsf{clean}_{\mathtt{EE}}$ implies the existence of a unique session at Bob with the same sid as that of Alice's session. As before, an adversary which could distinguish this game from its predecessor allows us to answer the Gap-DH challenge, violating that assumption. Indistinguishability of this stage's message key, as well as the next root and chaining keys enumerated in Case 4.4.1, then follows from applying the (random oracle) KDF to the (now independent) DH shared secret. Thus

$$\Pr(break_3) \leq 1/q + 1/2 + \epsilon_{\text{GDH}}$$

C4.2: Asymmetric ratcheting, $s = [\textbf{asym-ri:}x], x > 1$, *type*$[s] = \texttt{asym-ri}$

Now we come to the case of non-initial asymmetric stages of type `asym-ri`. The proof here is nearly the same as in Case 4.1, except there is an extra KDF application: session keys derived by these stages are computed by first applying a KDF to derive an intermediate value tmp, and second applying another KDF to derive from tmp a session key.

Similarly, we partition our analysis into the following cases.

- event $E^{\texttt{asym-ri}}_{\text{clean-prev}}$: $\mathsf{clean}_{\texttt{asym-ir}}(u, i, [\textbf{asym-ir:}x])$
- event $E^{\texttt{asym-ri}}_{\text{clean-cur}}$: $\mathsf{clean}_{\mathtt{EE}}(u, i, x, x-1)$

C4.2.1: Case $s = [\textbf{asym-ri:}x], x > 1$, *type*$[s] = \texttt{asym-ri}$ and $E^{\texttt{asym-ri}}_{\textbf{clean-prev}}$

Once again the inductive argument here is analogous to Case 3.2: secrecy follows from the root key, and so we begin by replacing the root key with a random value. Detecting this change would violate the security properties of the previous stage, but after it the session key is easily proven indistinguishable from random. This stage's message key $mk^{\textbf{sym-ri}:x,0}$ as well as symmetric chaining keys $ck^{\textbf{sym-ri}:x,0}$ and $ck^{\textbf{sym-ri}:x,1}$ and intermediate value tmp, are all also then indistinguishable from random. Thus

$$\Pr(break_3) = 1/q + \Pr(E^{\texttt{triple+DHE}}) + \epsilon_{\text{GDH}} + 1/2$$

C4.2.2: Case $s = [\textbf{asym-ri:}x], x > 1$, *type*$[s] = \texttt{asym-ri}$ and $E^{\texttt{asym-ri}}_{\textbf{clean-cur}}$

For this case, we proceed similarly to Case 3.1. The DH shared secret can be shown indistinguishable under the Gap-DH assumption by replacing the ratchet public keys $rchk_u^x, rchk_v^{x-1}$ of the Test session and its

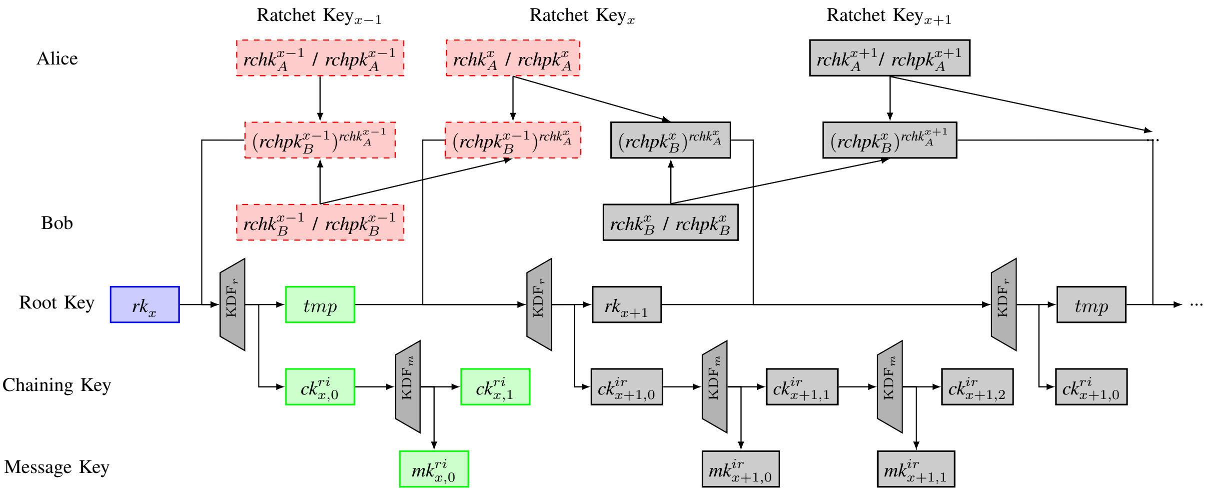


Figure 17: A diagram showing the replacement of secrets in Game 7' of Case 4.2.1. In particular, [red dashed box] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [green box] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [blue box] denotes secrets that the challenger is able to replace with random, thus ensuring security; [grey box] denotes secrets that are not relevant to this case; and [purple dashed box] denotes secrets one of which is assumed uncompromised but the rest may be revealed by $\mathcal{A}$.

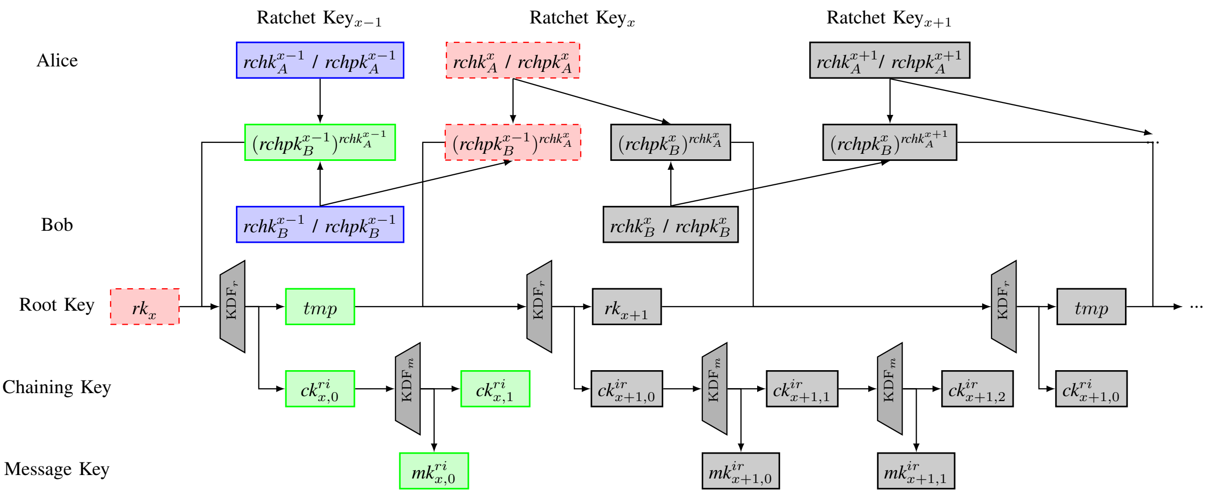


Figure 18: A diagram showing the replacement of secrets in Game 3 of Case 4.2.2. In particular, [red dashed box] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [green box] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [blue box] denotes secrets that the challenger is able to replace with random, thus ensuring security; [grey box] denotes secrets that are not relevant to this case; and [purple dashed box] denotes secrets one of which is assumed uncompromised but the rest may be revealed by $\mathcal{A}$.

matching peer with values from a GDH challenger. Indistinguishability of the stage's message key, symmetric chaining keys, and intermediate value tmp (as enumerated in case 4.2.1) all follow in turn from applying a (random oracle) KDF to (now independent) secret values. Thus

$$\Pr(break_3) \leq {}^1\!/\!_q + \epsilon_{\mathrm{GDH}} + {}^1\!/\!_2$$

C5: Symmetric ratcheting: *type*$[s] \in \{\texttt{sym-ir}, \texttt{sym-ri}\}$

We finally arrive at the case of Signal in the multistage key-indistinguishability game against an adversary $\mathcal{A}$ that issues a $\mathsf{Test}(u, i, [\textbf{sym-ir}{:}x{,}y])$ or $\mathsf{Test}(u, i, [\textbf{sym-ri}{:}x{,}y])$ query against some some symmetric stage. In all subcases, we will show that the probability of winning is ${}^1\!/\!_2$

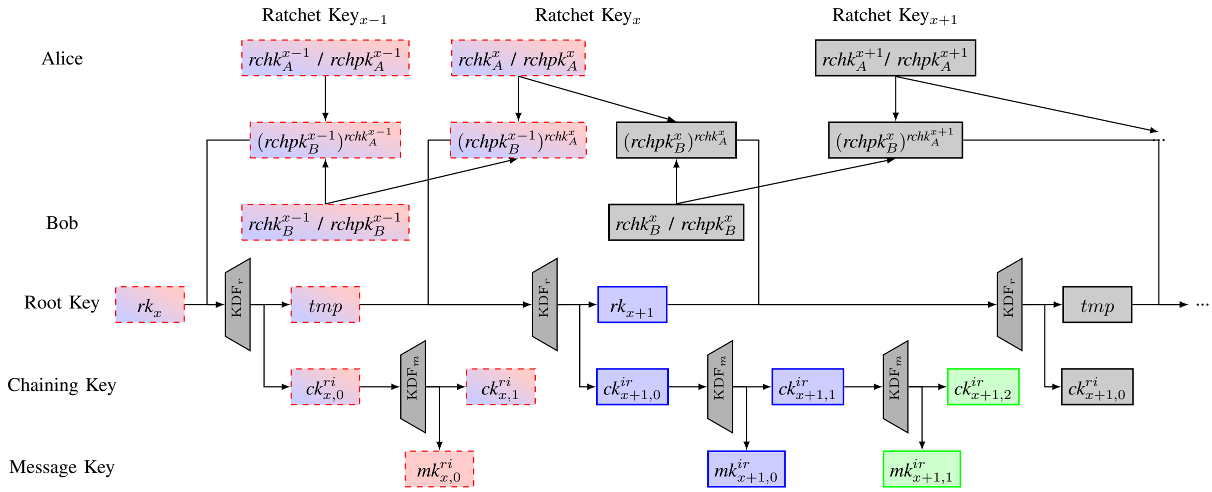


Figure 19: A diagram showing the replacement of secrets in Game 7' of Case 5.1.2s. All other subcases follow a similar replacement strategy. In particular, denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; denotes secrets that the challenger is able to replace with random, thus ensuring security; denotes secrets that are not relevant to this case; and denotes secrets one of which is assumed uncompromised but the rest may be revealed by $\mathcal{A}$.

C5.1: Symmetric ratcheting, $s = [\textbf{sym-ir}{:}x,y], \textbf{\textit{type}}[s] = \texttt{sym-ir}$

We partition into the three different freshness conditions for the case $[\textbf{sym-ir}{:}x,y]$. We then cover the case of $[\textbf{sym-ri}{:}x,y]$ similarly. The intuition is that for the first stage, the symmetric keys are derived from an asymmetric update and their secrecy follows from the previous cases. For subsequent stages, we have security due to the recursive nature of the freshness condition: we can replace the chain key used to derive the message key with randomness; if the simulation did not work, then the adversary could attack the previous stage, which is a contradiction to security of previous cases because the previous stage is fresh. In all symmetric stages, no new ephemeral keying material is introduced, so security depends solely on the chaining state not being leaked (which is guaranteed for these cases by $\mathsf{clean}_{\mathsf{state}}$).

(Recall that the case $y = 0$ is performed as part of the message key derivation in the previous asymmetric update, so that the first symmetric stage derives key number 1.)

C5.1.1: Case $s = [\textbf{sym-ir}{:}x,y], x = 0, y = 1, \textbf{\textit{type}}[s] = \texttt{sym-ir}$

This stage's messaging key is derived by applying a $\mathrm{KDF_m}$ to $ck^{\textbf{sym-ir}:0,1}$, which was derived during the initial `triple` or `triple+DHE` handshake. By Case 3.2, $ck^{\textbf{sym-ir}:0,1}$ is indistinguishable from random. Like the argument in Case 3.2, treating $\mathrm{KDF_m}$ as a random oracle, this stage's messaging key, as well as the next chaining key $ck^{\textbf{sym-ir}:0,2}$, are thus indistinguishable from random.

C5.1.2: Case $s = [\textbf{sym-ir}{:}x,y], x > 0, y = 1, \textbf{\textit{type}}[s] = \texttt{sym-ir}$

This stage's messaging key is derived by applying a $\mathrm{KDF_m}$ to $ck^{\textbf{sym-ir}:x,1}$, which was derived during asymmetric-second-stage $[\textbf{asym-ir}{:}x]$. By Case 4.1, $ck^{\textbf{sym-ir}:x,1}$ is indistinguishable from random. Like the argument in case C.2, treating $\mathrm{KDF_m}$ as a random oracle, this stage's messaging key, as well as the next chaining key $ck^{\textbf{sym-ir}:x,2}$, are thus indistinguishable from random.

C5.1.3: Case $s = [\textbf{sym-ir}{:}x,y], x \geq 0, y.1, \textbf{\textit{type}}[s] = \texttt{sym-ir}$

This stage's messaging key is derived by applying a $\mathrm{KDF_m}$ to $ck^{\textbf{sym-ir}:x,y}$, which was derived during symmetric stage $[\textbf{sym-ir}{:}x]y - 1$. By Case 5.1.1, 5.1.2, or induction on Case 5.1.3, $ck^{\textbf{sym-ir}:x,y-1}$ is indistinguishable from random. Like the argument in case 3.2, treating $\mathrm{KDF_m}$ as a random oracle, this stage's messaging key, as well as the next chaining key $ck^{\textbf{sym-ir}:x,y+1}$, are thus indistinguishable from random.

C5.2: Symmetric ratcheting, $s = [\textbf{sym-ri:}x{,}y], \textbf{\textit{type}}[s] = \texttt{sym-ri}$

These cases are analogous to Case 5.1, by symmetry: cleanness is defined in the same recursive manner for both `sym-ir` and `sym-ri` stages, except that the base cases differ. The initial game hops are thus analogous to those in the `asym-ir` and `asym-ri` respectively, and the subsequent inductive argument is analogous to Case 5.1.3. $\square$

Appendix C. Achieving a Standard Model proof of the Signal Protocol

One drawback of the proof as it currently stands is that it sits within the random oracle model. This is an assumption which has received some criticism from parts of the cryptographic community because, while it is a useful assumption for proofs, it can never be satisfied in reality [9, 37]. There are even pathological constructions which are provably secure with random oracles, but which can never be secure when the random oracle is replaced with any concrete primitive [5, 19]. While the use of the random oracle model does not imply an attack, and in fact sometimes deliberately avoiding it can cause more severe problems [45], its use may be unsettling for some.

Therefore, in this section, we outline the process in which one could attempt a standard model proof of the Signal Protocol in our security model. This is not a straightforward task; a naive attempt would be by retaining the current structure of our proof and replacing Gap-DH assumptions with a pair of DDH and PRF security assumptions. However, this approach does not fit. Similarly to previous proofs of TLS [30, 41], we find ourselves relying instead on the PRF-ODH assumption. There are many different variants of the PRF-ODH assumption throughout the literature; see [18] for a detailed exposition of these variants. However, the crux of the assumption is as follows. Given g^u and g^v, computing a function $\mathrm{PRF}(g^{uv}, x)$ should be hard, even with choice of x and an oracle that computes values like $\mathrm{PRF}(g^{uw}, x')$ and $\mathrm{PRF}(g^{wv}, x')$ for chosen $w \neq u, v$.

Roughly speaking, the reason a PRF-ODH assumption is required is that there are long-lived keys used in key computations in Signal that must be simulated when they are substituted out in the game hops. Before, we used a random oracle—which, by definition, gives random replies that the simulator could just choose randomly but consistently—to simulate these key computations. Now, we need a Diffie-Hellman oracle from the PRF-ODH assumption to carry out the simulation. Because Signal computes keys using a PRF on Diffie-Hellman values, it seems at least plausible that some version of a PRF-ODH assumption may work for a proof.

Readers may ask why we cannot simply retain the current structure of our proof and replace the Gap-DH assumptions with a single PRF-ODH step. As discussed in Section 2 (and wholly unlike TLS), the Diffie-Hellman values used in Signal can be long-term, medium-term or ephemeral. These keys are also used in different combinations in key derivations. This difference requires that we not use a single PRF-ODH assumption, but a range of PRF-ODH assumptions that are parameterized by how many times the challenger will generate $\mathrm{PRF}(g^{uv'}, x)$ and $\mathrm{PRF}(g^{u'v}, x)$ values upon being queried for a "left" or "right" PRF-ODH oracle $(g^{v'}, x)$ or $(g^{u'}, x)$ (where $g^{u'}$ and $g^{v'}$ are not one of the DH challenge values (g^u, g^v) given by the challenger). We give a formal definition below from the work by Brendel et. al. [18].

Definition 10 (Generic PRF-ODH assumption)**.** Let $\mathbb{G}$ be a cyclic group of order q with generator g. Let $\mathrm{PRF} : \mathbb{G} \times \{0,1\}^* \to \{0,1\}^\lambda$ be a pseudo-random function that takes as key input an element $k \in \mathbb{G}$ and an arbitrary-length salt value $x \in \{0,1\}^*$ as input, and outputs a value $y \in \{0,1\}^\lambda$.

We define a generalised security notion lr-PRF-ODH, which is parameterised by $l, r \in \{\mathsf{n}, \mathsf{s}, \mathsf{m}\}$, indicating how often the adversary is able to query a certain left oracle or right oracle (denoted ODH_u and ODH_v respectively) where n indicates that no query is allowed, s indicates that a single query is allowed, and m indicates that arbitrarily-many queries are allowed to the respective oracle. Consider the lr-PRF-ODH security experiment depicted in Figure 20.

We say that the adversary $\mathcal{A}$ wins the lr-PRF-ODH game if $b' = b$ and define the following advantage function:

$$\mathsf{Adv}^{lr\text{-}\mathsf{PRF\text{-}ODH}}_{\mathrm{PRF},\mathbb{G}}(\mathcal{A}) := |\Pr(\mathsf{Exp}^{lr\text{-}\mathsf{sPRF\text{-}ODH}}_{\mathrm{PRF},\mathbb{G},g,q}(\mathcal{A}) = 1) - \frac{1}{2}|$$

To add to the difficulty, these left-and-right generic PRF-ODH assumption variants do not allow the adversary to query both sides of the DH keyshares multiple times before the challenger generates the secret value, which would be the case in the replacement of the long-term and medium-term secrets (refer to Case 1.1), which means that we would need to further modify the generic PRF-ODH assumption to the needs of our particular Signal Protocol proof. We call this a *symmetric* generic PRF-ODH problem, which we define below.

Definition 11 (Symmetric PRF-ODH assumption)**.** Let $\mathbb{G}$ be a cyclic group of order q with generator g. Let $\mathrm{PRF} : \mathbb{G} \times \{0,1\}^* \to \{0,1\}^\lambda$ be a pseudo-random function that takes as key input an element $k \in \mathbb{G}$ and an arbitrary-length salt value $x \in \{0,1\}^*$ as input, and outputs a value $y \in \{0,1\}^\lambda$.

We define a symmetric security notion lr-sPRF-ODH, which is parameterised by $l, r \in \{\mathsf{n}, \mathsf{s}, \mathsf{m}\}$, indicating how often the adversary is able to query a certain left oracle or right oracle (denoted ODH_u and ODH_v

$\mathsf{Exp}^{lr\text{-}\mathsf{PRF\text{-}ODH}}_{\mathrm{PRF},\mathbb{G},g,q}(\mathcal{A})$:

$\mathsf{b} \xleftarrow{\$} \{0,1\}$, $c_u \leftarrow 0$, $c_v \leftarrow 0$
$u \xleftarrow{\$} \mathbb{Z}_q$, $\mathbf{X} \leftarrow \emptyset$
if $l = \mathsf{m}$ **then**
    $x^* \leftarrow \mathcal{A}(\mathbb{G}, g, q, g^u)^{\mathsf{ODH}_u}$
else
    $x^* \leftarrow \mathcal{A}(\mathbb{G}, g, q, g^u)$
$v \xleftarrow{\$} \mathbb{Z}_q$
if $\{g^{uv}, x^*\} \in \mathbf{X}$ **then**
    return $\perp$
$y_0 \leftarrow \mathrm{PRF}(g^{uv}, x^*)$, $y_1 \xleftarrow{\$} \{0,1\}^\lambda$
$\mathbf{X} \leftarrow \mathbf{X} \cup \{g^{uv}, x^*\}$
$b' \leftarrow \mathcal{A}(y_b)^{\mathsf{ODH}_u,\mathsf{ODH}_v}$
return $(b' = b)$

$\mathsf{Exp}^{lr\text{-}\mathsf{sPRF\text{-}ODH}}_{\mathrm{PRF},\mathbb{G},g,q}(\mathcal{A})$:

$\mathsf{b} \xleftarrow{\$} \{0,1\}$, $c_u \leftarrow 0$, $c_v \leftarrow 0$
$u, v \xleftarrow{\$} \mathbb{Z}_q$, $\mathbf{X} \leftarrow \emptyset$
if $(l = \mathsf{m}) \wedge (r = \mathsf{m})$ **then**
    $\mathcal{A}(\mathbb{G}, g, q, g^u, g^v)^{\mathsf{ODH}_u,\mathsf{ODH}_v} \rightarrow x^*$
if $(l = \mathsf{m}) \wedge (r \neq \mathsf{m})$ **then**
    $\mathcal{A}(\mathbb{G}, g, q, g^u, g^v)^{\mathsf{ODH}_u} \rightarrow x^*$
if $(l \neq \mathsf{m}) \wedge (r = \mathsf{m})$ **then**
    $\mathcal{A}(\mathbb{G}, g, q, g^u, g^v)^{\mathsf{ODH}_v} \rightarrow x^*$
else
    $\mathcal{A}(\mathbb{G}, g, q, g^u, g^v) \rightarrow x^*$
if $(g^{uv}, x^*) \in \mathbf{X}$ **then**
    return $\perp$
$y_0 \leftarrow \mathrm{PRF}(g^{uv}, x^*), y_1 \xleftarrow{\$} \{0,1\}^\lambda$
$\mathbf{X} \leftarrow \mathbf{X} \cup \{g^{uv}, x^*\}$
$b' \leftarrow \mathcal{A}(y_b)^{\mathsf{ODH}_u,\mathsf{ODH}_v}$
return $(b' = b)$

$\mathsf{ODH}_u(S, x, \mathbf{X}, c_u)$:

if $(l = \mathsf{n}) \vee (\{S^u, x\} \in \mathbf{X}) \vee (S \notin \mathbb{G})$ **then**
    return $\perp$
if $(l = \mathsf{s}) \wedge (c_u > 0)$ **then**
    return $\perp$
$c_u \leftarrow c_u + 1, \mathbf{X} \leftarrow \mathbf{X} \cup \{S^u, x\}$
return $\mathrm{PRF}(S^u, x)$

$\mathsf{ODH}_v(S, x, \mathbf{X}, c_v)$:

if $(r = \mathsf{n}) \vee (\{S^v, x\} \in \mathbf{X}) \vee (S \notin \mathbb{G})$ **then**
    return $\perp$
if $(r = \mathsf{s}) \wedge (c_v > 0)$ **then**
    return $\perp$
$c_v \leftarrow c_v + 1, \mathbf{X} \leftarrow \mathbf{X} \cup \{S^v, x\}$
return $\mathrm{PRF}(S^v, x)$

Figure 20: The security experiments for the generic PRF-ODH assumption, and the symmetric PRF-ODH assumption. Note that both experiments make use of identical ODH_u and ODH_v oracles.

respectively) where n indicates that no query is allowed, s indicates that a single query is allowed, and m indicates that polynomially-many queries are allowed to the respective oracle. Consider the security game $\mathsf{Exp}^{lr\text{-}\mathsf{sPRF\text{-}ODH}}_{\mathrm{PRF},\mathbb{G},g,q}(\mathcal{A})$ described in Figure 20. We say that the adversary $\mathcal{A}$ wins the lr-sPRF-ODH game if $b' = b$ and define the following advantage function:

$$\mathsf{Adv}^{lr\text{-}\mathsf{sPRF\text{-}ODH}}_{\mathrm{PRF},\mathbb{G},\mathcal{A}}(\lambda) := |\Pr(\mathsf{Exp}^{lr\text{-}\mathsf{sPRF\text{-}ODH}}_{\mathrm{PRF},\mathbb{G},g,q}(\mathcal{A}) = 1) - \frac{1}{2}|$$

However, Brendel et. al. [18] also show (via a algebraic reduction and meta-reduction argument) that the existence of any efficient black-box reduction from the sn/ns-PRF-ODH problem to a decisional Diffie-Hellman-augmented (DDHa) problem would imply that either the DDHa problem or the decisional-square Diffie-Hellman problem is not hard. The DDHa assumption is a class of assumptions, roughly stating that the adversary cannot efficiently win between either the decisional Diffie-Hellman problem or some other independent cryptographic problem. This, the authors argue, shows that the existence of a standard-model instantiation of any generic PRF-ODH problem (excepting nn-PRF-ODH) is impossible, assuming the aforementioned problems are indeed hard. So constructing a standard model proof of the Signal Protocol using generic PRF-ODH based assumptions could be moot.

There is however some advantage to this effort: it would bring clarity to which of the cases would be easier for the adversary to break. In the work by Brendel et. al. the relations and separations between the variants of lr-PRF-ODH are shown, and thus the security of each of the cases is able to be compared concretely. In our current proof, the adversary seemingly does not have an advantage targeting one particular case over another. Now we have all the tools we would require to consider how the security proof in each case would work. We consider each case below and explain which flavour of PRF-ODH is required and why.

Case 1.1 In Game 6 of Case 1.1 we know that the long-term identity key of party u and the medium-term key of the peer v have not been compromised by the adversary, and thus we can replace the key shares ipk_u, $prepk_v$ and the computed root key rk_1 and first-stage chain key $ck^{ir}_{0,0}$ with PRF-ODH challenge values. Since both of these Diffie-Hellman key shares can be used in multiple sessions, and (potentially) may be used before the Test session has initialised, we require many ODH_u and ODH_v queries at the start of the experiment before the challenge salt value x is computed. Thus we require the mm-sPRF-ODH assumption, the strongest variant of the PRF-ODH problem. In this case, we treat the keys ipk_u and $prepk_v$ as the keys to the PRF-ODH problem (which is now internal to the mm-sPRF-ODH game) and the following (potentially revealed) secrets ipk_v and ek_u values as the salt value x that is queried to the mm-sPRF-ODH challenger. The challenge value y_b output by the challenger is then used to replace the rk_1, $ck^{ir}_{0,0}$ values in both the Tested session and its peer session that is used in computing the message key $mk^{ir}_{0,0}$ which was Tested by the adversary. In order to ensure that the message key $mk^{ir}_{0,0}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk^{ir}_{0,0}$ from KDF_m, and use the output from the PRF challenger to replace $mk^{ir}_{0,0}$. Thus an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the mm-sPRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$.

Case 1.2 In Game 6 of Case 1.2 we know that the predicate $\mathsf{clean}_{\mathtt{EL}}(u, i, [0])$ is upheld, which means that the ephemeral key of the initiator and the identity-key of the responder have not been compromised by the adversary, and thus we can replace the key shares epk_u, ipk_v, and the computed root key rk_1 and first-stage chain key $ck^{ir}_{0,0}$ with PRF-ODH challenge values. Since only the identity-key of the responder can be used in multiple sessions, and (potentially) may be used before the Test session has initialised, we require many ODH_u queries at the start of the experiment before the challenge salt value x is computed. Thus we require the mn-PRF-ODH assumption, a non-symmetric variant of the PRF-ODH problem. In this case, we treat the values ipk_v, ek_u as the keys to the PRF-ODH problem (which is now internal to the mn-PRF-ODH game) and the following (potentially revealed) secrets $prepk_v$, ik_u as the salt value x that is queried to the mn-PRF-ODH challenger. The challenge value y_b output by the challenger is then used to replace the rk_1, $ck^{ir}_{0,0}$ values in both the Tested session and it's peer session that is used in computing the message key $mk^{ir}_{0,0}$ which was Tested by the adversary. In order to ensure that the message key $mk^{ir}_{0,0}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk^{ir}_{0,0}$ from KDF_m, and use the output from the PRF challenger to replace $mk^{ir}_{0,0}$. Thus an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the mn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$.

Case 1.3 In Game 6 of Case 1.3 we know that the predicate $\mathsf{clean}_{\mathtt{EM}}(u, i, [0])$ is upheld, which means that the ephemeral key of the initiator and the medium-term key of the responder have not been compromised by the adversary, and thus we can replace the keyshares epk_u, $prepk_v$ and the computed root key rk_1 and first-stage chain key $ck^{ir}_{0,0}$ with PRF-ODH challenge values. Since only the signed-prekey of the responder can be used in multiple sessions, and (potentially) may be used before the Test session has initialised, we require many ODH_v queries at the start of the experiment before the challenge salt value x is computed. Thus we require the mn-PRF-ODH assumption, a non-symmetric variant of the PRF-ODH problem. In this case, we treat the values $prepk_v$, ek_u as the key to the PRF-ODH problem (which is now internal to the mn-PRF-ODH game) and the following (potentially revealed) secrets ipk_v, ik_u values as the salt value x that is queried to the mn-PRF-ODH challenger. The challenge value y_b output by the challenger is then used to replace the rk_1, $ck^{ir}_{0,0}$ values in both the Tested session and it's peer session that is used in computing the message key $mk^{ir}_{0,0}$ which was Tested by the adversary. In order to ensure that the message key $mk^{ir}_{0,0}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk^{ir}_{0,0}$ from KDF_m, and use the output from the PRF challenger to replace $mk^{ir}_{0,0}$. Thus an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the mn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$.

Case 2.1 This is treated identically to Case 1.1, with the same bounds and game hops.

Case 2.2 This is treated identically to Case 1.2, with the same bounds and game hops.

Case 2.3 This is treated identically to Case 1.3, with the same bounds and game hops.

Case 2.4 In Game 6 of Case 2.4 we know that the predicate $\mathsf{clean}_{\mathtt{EE}}(u, i, [0])$ is upheld, which means that the ephemeral key of the initiator and the ephemeral key of the responder have not been compromised by the adversary, and thus we can replace the key shares epk_u, epk_v and the computed root key rk_1 and first-stage chain key $ck^{ir}_{0,0}$ with PRF-ODH challenge values. Since both keys are ephemerally generated and only used a single time, we require the sn-PRF-ODH assumption, the weakest non-standard model variant of the PRF-ODH problem. In this case, we treat the epk_v, ek_u as the key to the PRF-ODH problem (which is now internal to the sn-PRF-ODH game) and the following (potentially revealed) secrets $prepk_v$, ik_v and ik_u values as the salt value x that is queried to the sn-PRF-ODH challenger. The challenge value y_b output by the challenger is then used to replace the rk_1, $ck^{ir}_{0,0}$ values in both the Tested session and it's peer session that is used in computing the message key $mk^{ir}_{0,0}$ which was Tested by the adversary. In order to ensure that the message key $mk^{ir}_{0,0}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk^{ir}_{0,0}$ from KDF_m, and use the output from the PRF challenger to replace $mk^{ir}_{0,0}$. Thus an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the sn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$.

Case 3.1 In Game 6 of Case 3.1 we know that the predicate $\mathsf{clean}_{\mathtt{EE}}(u, i, [\mathbf{asym\text{-}ri}{:}1])$ is upheld, which means that the ratchet key of the initiator and the ratchet key of the responder have not been compromised by the adversary, and thus we can replace the key shares $rchpk_u^0$, $rchpk_v^0$ and the computed temporary value x and asymmetric-stage chain key $ck_{0,0}^{ri}$ with PRF-ODH challenge values. Since both keys are ephemerally generated and only used a single time, we require the sn-PRF-ODH assumption, the weakest non-standard model variant of the PRF-ODH problem. In this case, we treat the values $rchpk_v^0$, $rchk_u^0$ as the key to the PRF-ODH problem (which is now internal to the sn-PRF-ODH game) and the (potentially revealed) root key rk_1 of the first stage as the salt value x that is queried to the sn-PRF-ODH challenger. The challenge value y_b output by the challenger is then used to replace the x, $ck_{0,0}^{ri}$ values in both the Tested session and it's peer session that is used in computing the message key $mk_{0,0}^{ri}$ which was Tested by the adversary. In order to ensure that the message key $mk_{0,0}^{ri}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk_{0,0}^{ri}$ from KDF_m, and use the output from the PRF challenger to replace $mk_{0,0}^{ri}$. Thus an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the sn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$.

Case 3.2 In Game 6 of Case 3.2 we know that the predicate $\mathsf{clean}_{\pi_u^i.type[0]}(u, i, [0])$ is upheld, which means that initial key-exchange stage has some Diffie-Hellman key share pair that has not been corrupted, and that the adversary has not revealed the state linking the initial key-exchange to this stage. Depending on which clean predicate that was upheld in the first stage, the replacement of Diffie-Hellman values is done as in Case 1.1, Case 1.2, Case 1.3 or Case 2.4. We know from these case analysis that the root key rk_1 is indistinguishable from random, and thus we are able to replace this value with a random value $rk_{1'}$ and note that an adversary capable of distinguishing this change can break the security of Case 1.1, Case 1.2, Case 1.3 or Case 2.4. We then use PRF game hops in a standard way to replace the derivation of the x, $ck_{0,0}^{ri}$ values in both the Tested session and it's peer session that is used in computing the message key $mk_{0,0}^{ri}$ which was Tested by the adversary. In order to ensure that the message key $mk_{0,0}^{ri}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk_{0,0}^{ri}$ from KDF_m, and use the output from the PRF challenger to replace $mk_{0,0}^{ri}$. Thus an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the mm-sPRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$ (as in Case 1.1), or the mn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$ (as in Cases 1.2 and 1.3), or the sn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$ (as in Case 2.4).

Case 4.1.1 In Game 6 of Case 4.1.1 we know that the predicate $\mathsf{clean}_{\mathtt{asym\text{-}ri}}(u, i, [\textbf{asym-ri}{:}x-1])$ is upheld, which means that either:

- the previous stage's ratchet keys have not been compromised by the adversary (in which case analysis follows from Case 3.1)
- the previous stage's state has not been compromised by the adversary (in which case analysis follows from Case 3.2)

In a similar way, then, we follow those cases to replace the appropriate uncompromised Diffie-Hellman key shares with challenge values from a PRFODH game. Thus an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of KDF_m, or the mm-sPRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$ (as in Case 1.1), or the mn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$ (as in Cases 1.2 and 1.3), or finally the sn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$ (as in Cases 2.4 and 3.1).

Case 4.1.2 In Game 6 of Case 4.1.2 we know that the predicate $\mathsf{clean}_{\mathtt{EE}}(u, i, x-1, x-1)$ is upheld, which means that the previous stages ratchet keys have not been compromised by the adversary and analysis follows from Case 3.1, with the same bounds and game-hops. In particular, this means that the adversary's advantage in breaking the key-indistinguishability of the Tested session key is bound by the PRF security of KDF_r and KDF_m, or the sn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$.

Case 4.2.1 In Game 6 of Case 4.1.1 we know that the predicate $\mathsf{clean}_{\mathtt{asym\text{-}ri}}(u, i, [\textbf{asym-ir}{:}x-1])$ is upheld. Note that similarly to the proof of Case 4.2.1, this follows identically to Case 4.1.1 with an additionally application of a PRF game to account for the intermediate computation of a tmp value.

Case 4.2.2 In Game 6 of Case 4.2.2 we know that the predicate $\mathsf{clean}_{\mathtt{EE}}(u, i, x-1, x-1)$ is upheld, which means that the previous stages ratchet keys have not been compromised by the adversary and analysis follows identically to Case 4.1.2, with an additional PRF game to account for the intermediate computation of a tmp value. In particular, this means that the adversary's advantage in breaking the key-indistinguishability of the Tested session key is bound by the PRF security of KDF_r and KDF_m, or the sn-PRF-ODH security of $\mathrm{HKDF}, \mathbb{G}$.

Case 5 In this Case, and all subcases, analysis follows from Case 4, with additional PRF game hops to inductively replace chaining keys that (via the cleanness predicate $\mathsf{clean}_{\mathtt{sym}}$ have not been compromised by the adversary, and thus follows from the security of the appropriate asymmetric stage.

From this vantage point, we can now compare the cases concretely. For instance, it is clear that the adversary's advantage of breaking Case 1.1 (where the long-term identity key of the initiator and the medium-term signed prekey of the responder have not been compromised by the adversary) is quantitatively higher than the adversary's advantage in breaking Case 2.4 (where the ephemeral key of the initiator and the one-time prekey of the responder have not been trivially compromised by the adversary). This is due to the fact that Case

1.1 (and identically, Case 2.1) require the strong symmetric variant of PRFODH (i.e. mm-PRF-ODH) whereas Case 2.4 (and similarly, Case 3.1) require the weak non-symmetric varient of PRFODH (i.e. sn-PRF-ODH). Cases 1.2, 1.3, 2.2 and 2.3 sit between these two, requiring multiple queries an ODH_u oracle, but no queries to the ODH_v oracle, as it simulates a long-term Diffie-Hellman key and a single-use ephemeral Diffie-Hellman key using a mn-PRF-ODH challenger.

In addition, this supplemental proof also allows us to consider any future work that examines the computational hardness of the generic and symmetric PRF-ODH assumptions in relation to the security of the Signal protocol.

Appendix D.
Version History

V1.0 (2016-10-27) Original release.
V1.1 (2016-10-31) Minor updates.

- Clarified description of UKS attack in related work.
- Remarked that in the implementation, one-time keys can be updated after registration.
- Added reference to XEddsa signature specification and full/peer deniability.
- Removed references to header encryption and online key exchange, which are not part of Signal proper.

V2.0 (2017-10-09) Major updates.

- Added simplified protocol flow diagram to aid intuition.
- Clarified and further explained the freshness predicates.
- Clarified and further explained how our model relates to key indistinguishability and ACCE-like definitions.
- Made substantial clarifications and expansions to the proof.
- Added illustrative figures to the proof.
- Wrote Section C on the use of the PRF-ODH assumption to replace GapDH + ROM.