J Cryptol (2020) 33:1914–1983
https://doi.org/10.1007/s00145-020-09360-1

Journal of
CRYPTOLOGY


# A Formal Security Analysis of the Signal Messaging Protocol

Katriel Cohn-Gordon
Oxford, UK
me@katriel.co.uk

Cas Cremers
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
cremers@cispa.saarland

Benjamin Dowling
ETH Zürich, Zurich, Switzerland
benjamin.dowling@inf.ethz.ch

Luke Garratt
Cisco Systems, San Jose, USA
lgarratt@cisco.com

Douglas Stebila
University of Waterloo, Waterloo, Canada
dstebila@uwaterloo.ca

Communicated by Hugo Krawczyk

Received 8 November 2017 / Revised 4 June 2020
Online publication 23 September 2020

**Abstract.** The Signal protocol is a cryptographic messaging protocol that provides end-to-end encryption for instant messaging in WhatsApp, Wire, and Facebook Messenger among many others, serving well over 1 billion active users. Signal includes several uncommon security properties (such as "future secrecy" or "post-compromise security"), enabled by a technique called *ratcheting* in which session keys are updated with every message sent. We conduct a formal security analysis of Signal's initial extended triple Diffie–Hellman (X3DH) key agreement and Double Ratchet protocols as a multi-stage authenticated 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 as a multi-stage model where there can be a "tree" of stages, rather than just a sequence. We then prove the security of Signal's key exchange 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 foundation 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 plug-in 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, had a variety of flaws that seriously undermine its security [37].

### *The Signal Protocol*

While there has been a range of activity in end-to-end encryption for instant messaging [32,70], the most prominent development in this space has been the Signal messaging protocol, "a ratcheting forward secrecy protocol that works in synchronous and asynchronous messaging environments" [54,55]. 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 can be roughly divided into three types of stages:

- The initial key exchange, or X3DH (extended triple Diffie–Hellman) protocol [63], which combines long-term, medium-term and ephemeral Diffie–Hellman keys to produce a shared secret "root" value.
- An asymmetric ratchet stage [62], where users alternate in sending new ephemeral Diffie–Hellman keys in a ping-pong fashion with previously generated root keys to generate forward-secret chaining keys.
- A symmetric ratchet stage [62], where users take no additional entropy but instead use key derivation functions to ratchet forward chaining keys to create symmetric encryption keys.

Each message sent by a user is encrypted using a fresh message key, which attempts to provide a high degree of forward secrecy. The ping-pong pattern of new ephemeral Diffie–Hellman keys injects additionally entropy into this process, which is intended to continually achieve perfect forward secrecy as well as post-compromise security.

The Signal protocol, and in particular its ratcheting construction, has a relatively complex history. TextSecure [55] was a secure messaging app and the predecessor to Signal. It contained the first definition of Signal's "Double Ratchet", which effectively combines ideas from OTR's asymmetric ratchet and a *symmetric* ratchet (which applies a symmetric key derivation function to create a new key, but does not incorporate fresh Diffie–Hellman (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 WhatsApp [72], Wire [38], and Facebook Messenger [33], as well as a host of variants in "secure messaging" apps, including Silent Circle [57], Pond [52], and (via the OMEMO extension [69] to XMPP) Conversations [24] and ChatSecure [4].

### *Security of Signal*

One might have expected 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. However, this was not the case when it was released: when WhatsApp's Signal Protocol integration was announced in 2015, there was little Signal Protocol documentation available, and no in-depth security analysis. This was in stark contrast to the ongoing development of the next version of the Transport Layer Security protocol, TLS 1.3, which explicitly involved academic analysis in its development [14,26,30,43,47,53].

Since then, documentation has been introduced for both the X3DH initial key exchange protocol [63] and the Double Ratchet protocol [62], covering both the asymmetric and symmetric ratcheting stages of the Signal protocol.

Frosch et al. [35,36] had 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 previously 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

[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.

not directly fit into traditional analysis models. Second, some of its claimed properties have only been formalised relatively recently [23].

### 1.1. *Contributions*

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

Our paper focuses on the multi-stage AKE protocol aspect of Signal. Our formalism in Sect. 4 defines what a multi-stage AKE protocol is and a generic security experiment for multi-stage AKE protocols capturing session-key indistinguishability and then provides specialized freshness conditions that capture specific properties of Signal.

Compared to previous multi-stage AKE security models which involve a single sequence of stages within each session, our model allows for a *tree* of stages which model the various "chains" in Signal.

Our model remains generic for multi-stage AKE protocols as it is parameterized by freshness and cleanness conditions that allow the model to capture different security properties of session key indistinguishability. We proceed to provide cleanness conditions that are specialized to capture specific properties of Signal corresponding to its unique security goals. Among the interesting aspects of our model are the subtle differences between security properties of keys derived via symmetric and asymmetric ratcheting. In particular, the relationship between the initial key exchange and the subsequent ratcheting stages is partly why we chose to capture both the initial key exchange and the subsequent protocol in a single model. In addition, there is no clear delineation within Signal between these two protocols, as some secret values generated during the initial X3DH key exchange are reused within the subsequent Double Ratchet protocol.

We subsequently prove that the key establishment core of Signal is secure in our model, providing the first formal security guarantees for Signal.

We give a proof sketch in Sect. 5 and the full proof in Sect. B.3. At a high level, the proof technique is relatively straightforward and conventional. We suppose that there exists an adversary which breaks Signal and construct an adversary which breaks a primitive. To do so, we perform a series of game hops, followed by a case distinction on the type of attack the adversary performs. Specifically, we split our analysis into five general cases, each corresponding to a "stage" that derives a message key, be they the initial X3DH key exchange, an asymmetric ratcheting stage, or a symmetric ratcheting stage. Within these stages, we split our analysis further into subcases, each corresponding to a pair of Diffie–Hellman keyshares that the attacker has not accessed the associated secret value of. After some more game hops, we arrive at a game which is unwinnable by construction. Combining the intermediate probability bounds leads us to our overall security theorem.

Signal does not cleanly separate key exchange from subsequent data messages, and so we had to rearrange the order of certain operations in order to make this separation. It also reuses Diffie–Hellman keys for signatures; we did not model this reuse. The details of these changes are described later.

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 Sect. 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 Sect. 2 can serve as a starting point for understanding Signal's core.

### 1.2. *Additional Related Work*

A moderate body of research has arisen around the Signal Protocol, of which we give a brief summary.

In concurrent work to the conference version of this paper, Kobeissi, Bhargavan, and Blanchet [45] use ProVerif and CryptoVerif to analyse a simplified variant of Signal specified in a JavaScript variant called ProScript. Their main focus is to present a methodology for automated verification for secure messaging protocols and their implementations, and they 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 Sect. 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.

#### *Related Techniques*

Symmetric ratcheting and Diffie–Hellman (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], has been claimed to offer properties such as "future secrecy". Future secrecy of protocols like Signal has been discussed in depth by Cohn-Gordon, Cremers, and Garratt [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 of Signal may depend on subtle details related to device state reset and the handling of multiple devices.

Green and Miers [40] 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.

Rösler, Mainka, and Schwenk [68] study practical vulnerabilities in the *group* chat implementations in a number of messaging systems. Our work, in contrast, focuses only on the two-party case.

*Secure Channel Models*

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.

Building on Bellare et al. [10], Poettering and Rösler [64] extend to a model which covers bidirectional updates, following a purist approach. They give a stronger notion of compromise than ours and give constructions meeting their definition from hierarchical identity-based encryption. Durak and Vaudenay [31] and Jost, Maurer, and Mularczyk [44] continue along these lines, the former avoiding key-update primitives and the latter slightly weakening the security definition while remaining stronger than ours. Finally, Alwen, Coretti, and Dodis [1] give a clean-slate security model which additionally captures "immediate decryption", which we do not cover in our analysis.

*Overview*

In Sect. 2, we give a detailed presentation of the Signal protocol. We follow this by a high-level description of its threat model in Sect. 3 and a formal security model in Sect. 4. In Sect. 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 Sect. 6. We conclude in Sect. 7.

## 2. The Core Signal Protocol

The Signal protocol aims to send encrypted messages from one party to another. At a high level, Signal is an asynchronous secure channel protocol, with keys computed by a multi-stage AKE protocol, between an initiator Alice and a responder Bob, with the help

of a key distribution server which only stores and relays information between parties, but does not perform any computation.

As stated above, we focus on solely the key establishment component of Signal, which can be thought of as a multi-stage authenticated key exchange protocol.

### *Basic Setup*

Signal 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 Diffie–Hellman (DH) key-exchange to achieve perfect-forward secrecy requires both parties to contribute new ephemeral Diffie–Hellman (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.

### *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] [54]. After the conference version of this paper was published, Open Whisper Systems released high-level specifications for X3DH [63] and the Double Ratchet [62] which help to clarify many details, although the codebase is still necessary to obtain a full definition and the specification does not contain detailed definitions of the security goals.

## 2.1. *Protocol Overview*

Focusing on Signal as a multi-stage authenticated key exchange protocol, the main steps are as follows:

[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.

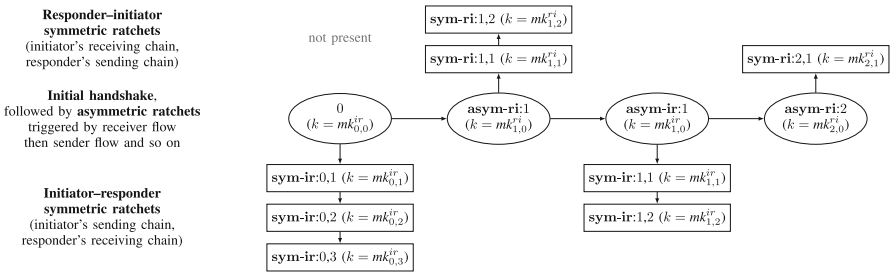



**Fig. 1.** A tree of *stages* from one example execution of Signal. The content of each node is the stage name and the session key derived during the stage .

**Registration.** (Section 2.6)
At installation (and periodically afterwards), both Alice and Bob independently register their identity with a key distribution server and upload some long-term, medium-term, and ephemeral public keys.
**Session setup.** (Section 2.7)
Alice requests and receives a set of Bob's public keys from the key distribution server and use them to set up a long-lived messaging session and establish initial symmetric encryption keys. This is called the TripleDH handshake or X3DH.
**Synchronous messaging (a.k.a. asymmetric-ratchet updates).** (Section 2.9)

When Alice wants to send a message to Bob (or vice versa) and has just received a message from Bob, she exchanges Diffie–Hellman values with Bob generating new shared secrets and uses them to begin new chains of message keys. Each Diffie–Hellman (DH) operation is a stage of the "asymmetric ratchet" (and strictly occurs in a ping-pong fashion).
**Asynchronous messaging (a.k.a. symmetric-ratchet).** (Section 2.8)

When Alice wants to send a message to Bob (or vice versa) but has not received a message from Bob since her last sent message to Bob, she derives a new symmetric encryption key from her previous state using a PRF. Each PRF application is a stage of the "symmetric ratchet".
Figure 1 shows a tree of stages from one example execution of Signal, represented as a graph. The session setup is the stage labelled 0. The asymmetric-ratchet update stages are labelled **asym-ri**:1, **asym-ir**:1, and so on, depending on whether a reply from the responder (**ri**) or initiator (**ir**) triggers the stage. The symmetric-ratchet stages are labelled **sym-ir**:0, 1, **sym-ir**:0, 2, **sym-ri**:1, 1, and so on, depending on whether they are derived from an initiator-triggered asymmetric stage (**ir**) or a responder-triggered asymmetric stage (**ri**). Each node in the graph in Fig. 1 shows a session key $k$ derived by the key exchange at each stage, which is the *message key mk* used to encrypt the application messages sent during that stage.

We call this entire execution a single *session* of a multi-stage AKE protocol. In fact, Alice and Bob can run many simultaneous sessions between them or with other parties, each admitting an arbitrary sequence of stages consisting of symmetric and asymmetric ratcheting.

Starting to look at the protocol messages, Fig. 2 shows a simplified form of the message flow for an example execution of the Signal protocol. (Simplifications include

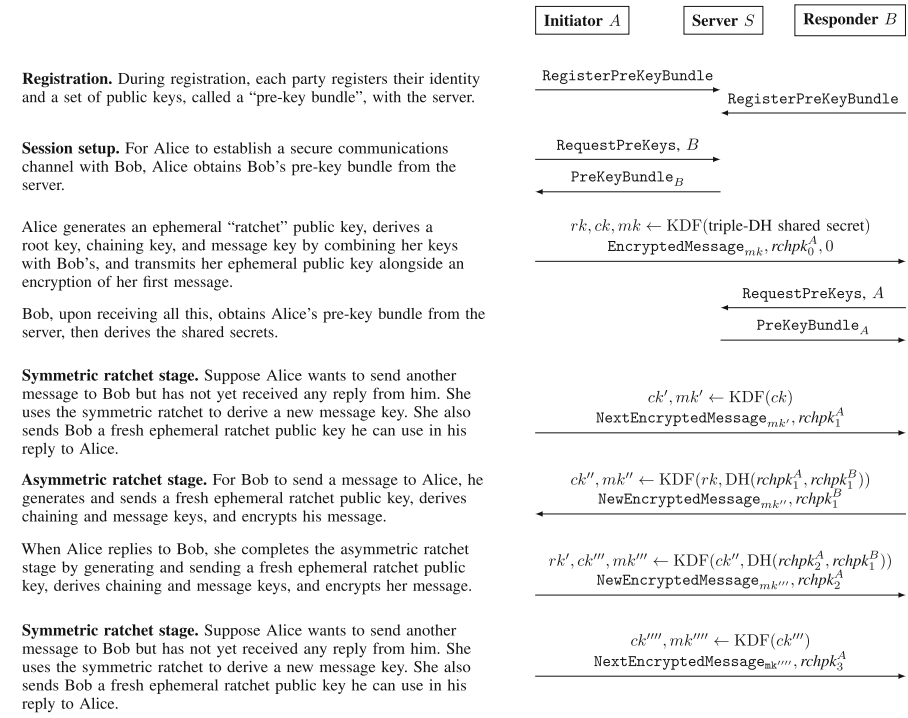



**Fig. 2.** Message flow of an example Signal execution between two clients $A$ and $B$ via a server $S$. Notation and some operations have been simplified for clarity compared to later use.

omission of some intermediate values and collapsing several KDF applications into a single application. A detailed version is given in Fig. 6.) Already we can see a few additional characteristics. At each asymmetric ratchet stage, the sending party sends a fresh ephemeral Diffie–Hellman public key. Each asymmetric ratchet stage derives its keying material from a DH pair (consisting the sending party's new ephemeral DH public key and the receiving party's previous ephemeral DH public key) and the *root key rk* from the previous asymmetric ratchet stage (or the initial handshake). Each symmetric ratchet stage derives its keying material from the *chaining key ck* from the previous symmetric stage (or the asymmetric stage which started this symmetric ratchet).

In the rest of this section, we state the cryptographic core of the Signal protocol that we analyse. We begin with some notation, including a list of the various keys used in the protocol, and then examine how each stage of the protocol works.

### 2.2. *Notation—Cryptographic Primitives*

Let $g$ denote the generator of a group $G$ of prime order $q$; we write the group multiplicatively. Signal uses one of two elliptic curves to implement X3DH: curve X25519 [12] or curve X448 [41].

Signal uses key derivation functions in two different ways, as shown in Fig. 7, applying either HMAC − SHA256 [6] or HKDF [48] using SHA256 as indicated.

AEAD denotes an authenticated encryption scheme with associated data [67]. In Signal, this is an encrypt-then-MAC scheme: encryption is AES256 in CBC mode with PKCS#5 padding, and the MAC is HMAC − SHA256. This is the same combination originally used in TextSecure v3, which was shown by Frosch et al. [36] 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 a signature scheme based on Ed25519 [13,61]. We treat it as a black-box signature.

### 2.3. *Notation—Sessions and Stages*

We denote $A$'s $i$th session by $\pi_A^i$. Note that a session refers to an execution of a protocol at a party, so in the normal run of the protocol between Alice and Bob, there will be two sessions: one at Alice and one at Bob.

Within a session, Signal admits a tree of various different stages, as shown in Fig. 1. We refer to stages using a term in [square brackets]. Each stage corresponds to the establishment of a new message encryption key.

The initial stage is [0]; the initiator retrieves the responder's prekey bundle of public keys from the key distribution server and then sends a flow to the responder.

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).

The asymmetric ratchet chain builds from the initial stage using alternating stages of two types. First, there is a flow from the responder to the initiator, denoted [**asym-ri**:1]. Upon receiving this flow, the initiator completes the asymmetric ratchet in a stage denoted [**asym-ir**:1]. They continue to alternate: [**asym-ri**:2], [**asym-ir**:2], and so on.

Symmetric ratchet chains are built from each stage of the asymmetric ratchet chain (including the initial stage), any time a party wants to send an additional application message before having received a response. Symmetric ratchet chain stages built from [**asym-ri**:$x$], for $x \geq 1$, are denoted [**sym-ri**:$x$, 1], [**sym-ri**:$x$, 2], and so on. Symmetric ratchet chain stages [**sym-ir**:$x$, 1], [**sym-ir**:$x$, 2], and so on, are built from [**asym-ir**:$x$], for $x \geq 1$, and [0] for $x = 0$.

### 2.4. *Notation—Keys*

Signal distinguishes between at least ten different classes of key, which are summarized in Table 1.

Our convention is that 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:

**Table 1.** Keys used in the signal protocol .

| | | |
|---|---|---|
| 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 (for the initial handshake) |
| $epk^A$ | $ek^A$ | $A$'s ephemeral key pair (for the initial handshake) |
| $rchpk_x^A$ | $rchk_x^A$ | $A$'s $x$th ratchet key pair (for the $x$th asymmetric ratchet) |
| Symmetric | | |
| | $ck_{x,y}^{ir}$ | $y$th chaining key in the $x$th initiator–responder symmetric ratchet chain |
| | $ck_{x,y}^{ri}$ | $y$th chaining key in the $x$th responder–initiator symmetric ratchet chain |
| | $mk_{x,y}^{ir}$ | $y$th message key in the $x$th initiator–responder symmetric ratchet chain |
| | $mk_{x,y}^{ri}$ | $y$th message key in the $x$th responder–initiator symmetric ratchet chain |
| | $rk_x$ | $x$th root key on the asymmetric ratchet |

Asymmetric key pairs show public and private components

$pk = g^k$. If the identity of the agent $A$ who generates a key is unclear, we mark this in superscript (i.e. $k^A$), but omit this where it is clear.

Every stage derives new keys. To identify these keys uniquely, we write the type of the stage deriving a key $k$ in superscript, and the index of that stage in subscript. For example, $mk_{x,y}^{ri}$ is the message key derived in stage [**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.

For the purposes of modelling key exchange, the "session key" of each stage is the message key derived during that stage. The exact mapping of session keys to stages is shown in Table 2.

The naming scheme for keys is also role-agnostic: in intended operation, keys will be equal if and only if 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 $mk_{x,y}^{ir}$ 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 [**sym-ir**:$x$, 0] or [**sym-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 Fig. 1). 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.

### 2.5. *The Protocol*

We now proceed to describe each stage of the protocol in detail. The full description of the protocol is given in Fig. 6, but we deliberately do not show that figure right away, instead using alternative diagrams which we hope provide greater clarity for specific parts of the protocol. However, each stage in the subsequent subsections corresponds to the portion of Fig. 6 as indicated.

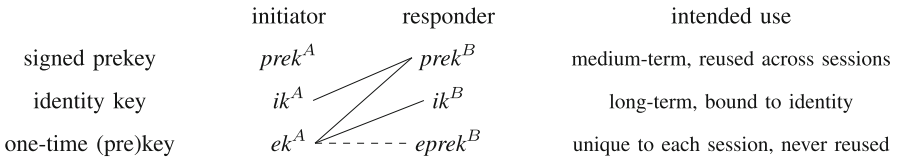



**Fig. 3.** Diffie–Hellman keys used in the initial handshake. The dashed line is optional: it is omitted from the session key derivation if $eprek^B$ is not sent. Note the asymmetry: when Alice initiates a session with Bob, her signed prekey is not used at all. Our freshness conditions in Sect. 4.3 will be partially based on this graph.

### 2.6. *Registration Stage—Figure 6a*

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

Specifically, each agent $P$ generates the following Diffie–Hellman (DH) private keys:

(i) a long-term "identity" key $ik^P$;
(ii) a medium-term "signed prekey" $prek^P$; and
(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*. These are collectively called the "prekey bundle".

### 2.7. *Session Setup Stage—Figure 6b*

In the session-setup stage, public keys are exchanged and used to initialize shared secrets. The underlying key exchange protocol is a one-round Diffie–Hellman (DH) protocol called the Signal Key Exchange or X3DH,[3] comprising an exchange of various Diffie–Hellman (DH) public keys, computation of various Diffie–Hellman (DH) shared secrets as in Fig. 3, and then application of a key derivation function.

While many possible variants of such protocols have been explored in-depth in the literature (HMQV [49], Kudla-Paterson [50], NAXOS [51] 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 [50].

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 Diffie–Hellman (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

[3] The key exchange protocol was previously referred to as TripleDH, from the three Diffie–Hellman (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 Diffie–Hellman (DH) value, not as might be expected the variant which includes the one-time prekey.

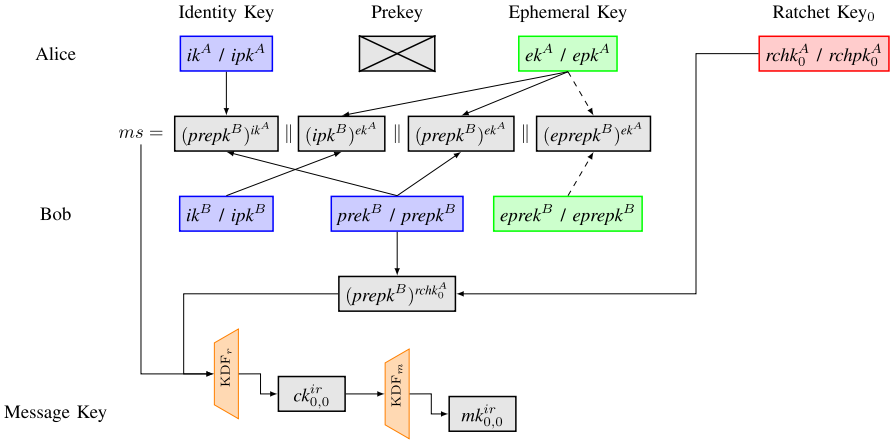



**Fig. 4.** A key schedule diagram for the initial X3DH key exchange of the Signal protocol. In this caption, the terminology "sending" and "receiving" refers to Alice's point of view. Note that ▭ denotes Diffie–Hellman (DH) values that are used in multiple X3DH exchanges, ▭ denotes Diffie–Hellman (DH) values that are used only in a single X3DH exchange, and ▭ denotes Diffie–Hellman (DH) values that are also used in the Double Ratchet protocol. This example key schedule captures an initial X3DH key exchange, in which Alice uses the standard cryptographic data from Bob's prekey bundle (plus a one-time ephemeral prekey $eprepk^B$) to compute the first root key $rk_1$, the first sending chain key $ck_{0,1}^{ir}$, and the first message key $mk_{0,0}^{ir}$ (Color figure online).

the received values as the first message of a Signal key exchange and completes the exchange in order to derive a master secret.

### 2.7.1. *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 precomputed 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 prekeys 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.

### 2.7.2. *Computing the Shared Secrets*

Figure 4 shows the key schedule for the initial handshake in greater detail. (This is a subset of Fig. 5.)

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 Fig. 3. She then concatenates the resulting shared secrets and passes them through a key derivation function ($\text{KDF}_\text{r}$, see Fig. 7a) to derive an initial root key $rk_1$ and sending chain key $ck_{0,0}^{ir}$. (No Diffie–Hellman (DH) value is passed to $\text{KDF}_\text{r}$ for this initial invocation.) For modelling purposes, we also have Alice generate her initial

sending message key $mk^{ir}_{0,0}$ (which is this stage's session key output) and the next sending chain key $ck^{ri}_{1,0}$. Finally, she generates a new ephemeral Diffie–Hellman (DH) key $rchk^A_0$ known as her ratchet key.

For Bob to complete[4] the key exchange, he must receive Alice's public ephemeral key $epk^A$ and public ratchet key $rchpk^A_0$. In the Signal protocol, Alice attaches these values to all messages that she sends (until she receives a message from Bob, since from such a message she can conclude that Bob received $epk^A$ and $rchpk^A_0$). To disentangle the stages of the model, we have Alice send $epk^A$, $rchpk^A_0$ 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 $epk^A$ and $rchpk^A_0$, he first checks that he currently knows the private keys corresponding to the identity, signed pre-, and one-time prekey 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 $ck^{ir}_{0,0}$). For modelling purposes, we also have Bob generate his initial receiving message key $mk^{ir}_{0,0}$ (which is this stage's session key output) and the next receiving chain key $ck^{ir}_{0,1}$.

The specific key derivation functions used are shown in Fig. 7.

### 2.8. *Symmetric-Ratchet Stage—Figure 6c*

Two sequences of symmetric keys will be derived using a PRF chain, one for sending and one for receiving. The symmetric chains—to the top and the bottom in Fig. 1—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 $ck^{ir}_{x,y}$ and applies the message key derivation function $\mathrm{KDF_m}$ to derive two new keys: an updated sending chain key $ck^{ir}_{x,(y+1)}$ and a sending message key $mk^{ir}_{x,y}$. Alice uses the sending message key to encrypt her outgoing message and then deletes it and the old sending chain key. This process can be repeated arbitrarily.

An example of the symmetric ratchet is shown in the bottom of Fig. 5. Specifically, on the lines labelled "chaining key" and "message key", there is one symmetric ratchet stage shown: stage [**sym-ir**:0, 1] which computes $mk^{ir}_{0,1}$.

When Alice receives an encrypted message, she checks the accompanying ratchet public key to confirm that she has not yet processed it, and if not, she then performs an asymmetric ratchet update, described below. Regardless, she then reads the metadata in the message header to determine the index of the message in the receiving chain and advances the receiving chain as many steps as are 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.)

[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.

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 the 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 core concepts of the root key producing chain keys, and the chain keys producing message keys, mean that messages can arrive in arbitrary order, while Alice and Bob can continue to asymmetrically and symmetrically ratchet forward.

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.

### 2.9. *Asymmetric-Ratchet Stage—Figure 6d*

The final top-level stage of Signal is the asymmetric-ratchet update. In this stage, Alice and Bob take turns generating and sending new Diffie–Hellman (DH) ratchet public keys and using them to derive new shared secrets. This behaviour occurs strictly in a ping-pong fashion, i.e. Bob will continue to use the same ratchet key until he sees a new ratchet key from Alice. In Fig. 6d, these newly generated keys are denoted by $rchpk_x^i$, where $i$ refers to the identity of the sending party (either $A$ or $B$ for Alice and Bob respectively) and $x$ refers to the numbered asymmetric stage i.e. [**asym-ir**:$x$] or [**asym-ri**:$x$]. 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 $rchpk_{x-1}^B$. If so, this triggers Alice to enter her next asymmetric ratchet stage [**asym-ir**:$x$]. Note that Alice already has stored a previously generated private ratchet key $rchk_{x-1}^A$. Before decrypting the message, Alice updates her asymmetric ratchet as per Fig. 6d. This consists of two steps:

- In the first step, denoted [**asym-ri**:$x$] in Fig. 6d, Alice derives three secret values: an intermediate secret value $tmp$, a receiving chain key $ck_{x,1}^{ri}$, and a receiving message key $mk_{x,0}^{ri}$. Alice computes first a Diffie–Hellman (DH) shared secret (between the newly received ratchet public key $rchpk_{x-1}^B$ and her old ratchet private key $rchk_{x-1}^A$) and combines this with the previously computed root chain key $rk_x$ to derive an intermediate secret value $tmp$ and a new receiving chain key $ck_{x,0}^{ri}$ by applying $\text{KDF}_r$. Afterwards, Alice uses the new chain key $ck_{x,0}^{ri}$ as input to $\text{KDF}_m$ to compute the next receiving chain key $ck_{x,1}^{ri}$ and receiving message key $mk_{x,0}^{ri}$. Alice generates a new Diffie–Hellman (DH) ratchet private key $rchk_x^A$.
- In the second step, denoted [**asym-ir**:$x$] in Fig. 6d, Alice computes a second Diffie–Hellman (DH) shared secret (between the received ratchet public key $rchpk_{x-1}^B$ and her new ratchet private key $rchk_x^A$) and combines this with the intermediate secret value $tmp$ by applying $\text{KDF}_r$ to compute the next root chain key $rk_{x+1}$ and a new

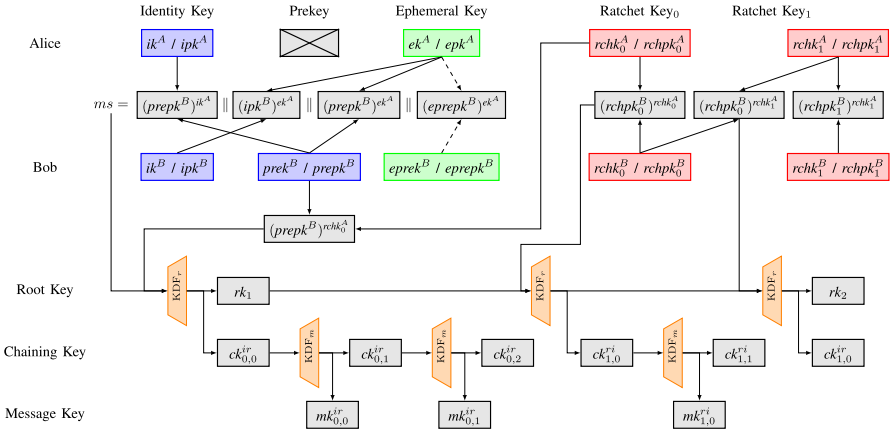



**Fig. 5.** A key schedule diagram for an example session of the Signal Protocol. In this caption, the terminology "sending" and "receiving" refers to Alice's point of view. Note that ▭ denotes Diffie–Hellman (DH) values that are used in multiple X3DH exchanges, ▭ denotes Diffie–Hellman (DH) values that are used only in a single X3DH exchange, and ▭ denotes public keys that are used in two asymmetric ratchets in a single Double Ratchet protocol. This example key schedule captures an initial X3DH key exchange, in which Alice uses the standard cryptographic data from Bob's prekey bundle (plus a one-time ephemeral prekey $eprepk^B$) to compute the first root key $rk_1$, the first sending chain key $ck^{ir}_{0,1}$, and the first message key $mk^{ir}_{0,0}$. Additionally, Alice has sent another message before receiving a message from Bob, which requires a *symmetric* ratchet of the first sending chain to derive the next message key $mk[ir][0,1]$. Afterwards, Bob has sent a new ratchet public key $rchpk^B_0$ that he combines with Alice's ratchet key $rchpk^A_0$ to compute the first receiving chain key $ck^{ri}_{1,0}$, used to derive the next chain key $ck^{ri}_{1,1}$ and the first receiving message key $mk^{ri}_{1,0}$. Finally, Alice has sent a new ratchet key $rchpk^A_1$ that is combined with Bob's previous ratchet public key $rchpk^B_0$ to compute the next root key $rk_2$ (Color figure online).

sending chain key $ck^{ir}_{x,0}$. Alice then uses the new sending chain key as input to $\mathrm{KDF}_m$ to generate the next sending chain key $ck^{ir}_{x,1}$ and receiving message key $mk^{ir}_{x,0}$.

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 [**asym-ri**:$x$] and [**asym-ir**:$x$].

Alice then sends her new ratchet public key $rchpk^A_x$ along with future messages to Bob, and the process continues indefinitely. An example key schedule diagram can be found in Fig. 5.

Figure 5 shows the key schedule for multiple stages of an example session of the Signal protocol. Two examples of the asymmetric ratchet are shown in the right of Fig. 5: stage [**asym-ri**:1] which computes $mk^{ri}_{1,0}$, and part of stage [**asym-ir**:1] which computes $mk^{ir}_{1,0}$ from $ck^{ir}_{1,0}$ (the computation of $mk^{ir}_{1,0}$ did not fit in the figure, but indeed is the output of that stage).

Bob does the corresponding operations shown in Fig. 6 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 Fig. 6c or its horizontal flip, asymmetric updates can only be

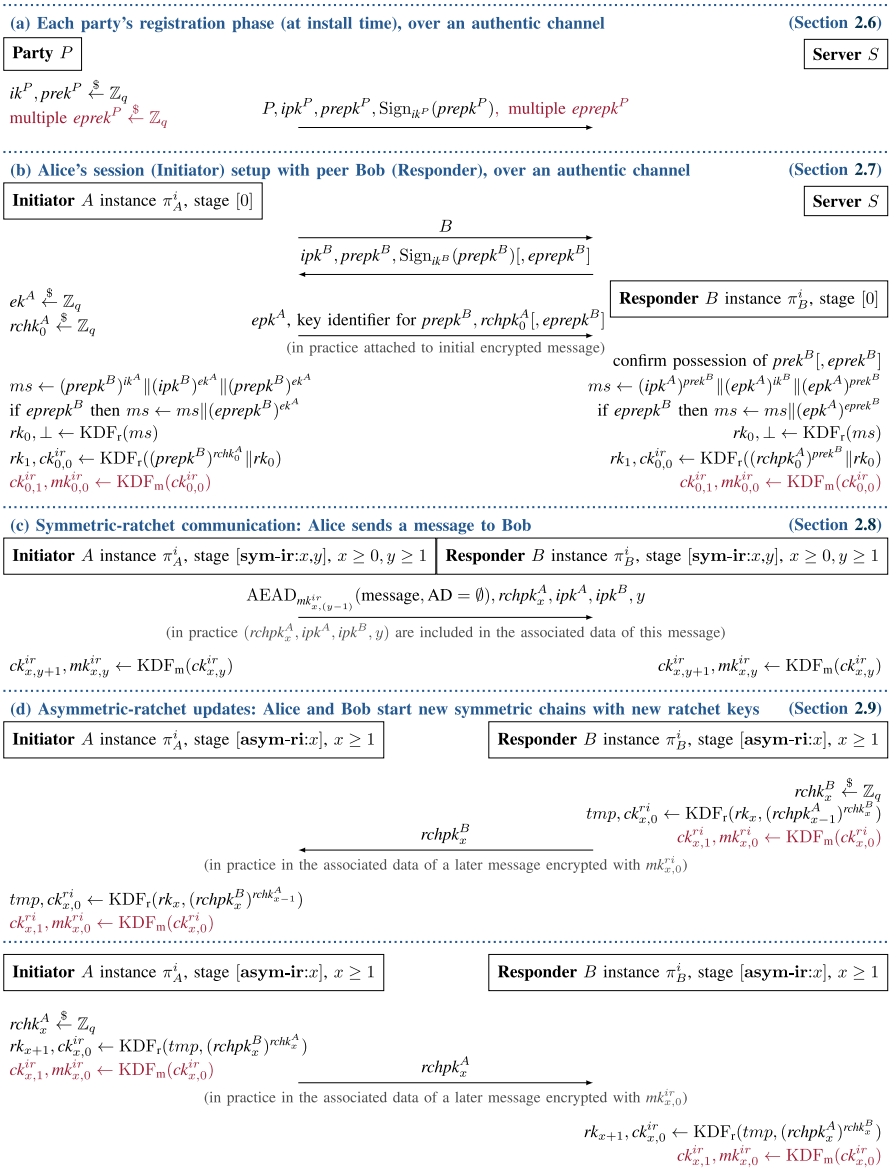
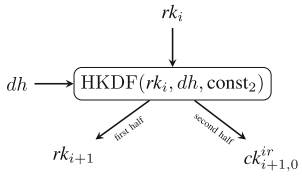
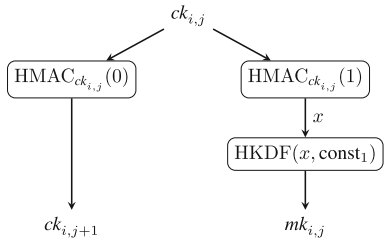



**Fig. 6.** Signal protocol including preregistration of keys. Local actions are depicted in the left and right columns, and messages flow between them. We show only one step of the symmetric and asymmetric ratchets; they can be iterated arbitrarily. Variables storing keys are defined in Table 1, $\mathrm{KDF_r}$ and $\mathrm{KDF_m}$ in Fig. 7, and session identifiers in Table 2. Dark red text indicates reordered actions in our model, as discussed in Sect. 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 (Color figure online).




(a) $\mathrm{KDF_r}$, the KDF for asymmetric ratchet 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.




(b) $\mathrm{KDF_m}$, the KDF for symmetric ratchet chain updates.

**Fig. 7.** Key derivation functions for root keys, chaining keys, and message keys in signal.

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 Fig. 6d will never be horizontally flipped. This means that the transmission of new ratchet keys between Alice and Bob will always occur in a strictly ping-pong fashion.

### 2.10. *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.

In Sect. 4, we will need to address specific values in the memory of specific sections, which are detailed in Table 2.

## 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 Sect. 4.3, 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 [54] 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 [66, 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.
> **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 $k$.
> **Key confinement** Distinct exchanges should derive distinct keys.
> **(Implicit) Authentication** If Alice believes that she shares the key $k$ with Bob, nobody other than Bob should be able to learn anything about $k$.
> **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 [71] in Signal, but these are relatively abstract. In general, signatures are used but only for the signed prekey 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 Diffie–Hellman (DH) shared secret is

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

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 secrets 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 [65], 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 6 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) [42]. It is often preferable to analyse 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 Fig. 6, 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 in the clear, with authentication enforced via our freshness predicate. We discuss these modifications further in Sect. 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 Fig. 1.

*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 uppercase letters; variables start with lowercase letters. We use object-oriented notation to represent collections of variables. In particular, we will use $\pi_u^i$ to denote the collection of variables that party $u$ uses in its $i^{\text{th}}$ protocol execution ("session"). To denote the variable $v$ in stage $s$ of party $u$'s $i$th session, we write $\pi_u^i.v[s]$; note $s$ is not (necessarily) a natural number. For Signal, $s$ is [0] for the session setup stage; [**sym-ir**:$x$, $y$] or [**sym-ri**:$x$, $y$] for symmetric sending or receiving stages; or [**asym-ri**:$x$] or [**asym-ir**:$x$] for the first and second portions of the $x$th asymmetric stage. (See also Figs. 1 and 6.)

Diffie–Hellman (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 Diffie–Hellman (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 Fig. 8, 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 the 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 Diffie–Hellman (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* $\Pi$ is a tuple of algorithms, along with a keyspace $\mathcal{K}$ and a security parameter $\lambda$ indicating the number of bits of randomness each session requires. The algorithms are:

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

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

- $\pi.role \in \{\texttt{init}, \texttt{resp}\}$: the instance's role
- $\pi.peerid$: the identifier of the alleged peer
- $\pi.peeripk$: the peer's long-term public key
- $\pi.peerprepk$: the peer's medium-term public key
- $\pi.status[s] \in \{\varepsilon, \texttt{active}, \texttt{accept}, \texttt{reject}\}$: execution status for stage $s$, set to `active` upon start of a new stage, and set to `accept` or `reject` by computation of the stage's ratchet key.
- $\pi.k[s] \in \mathcal{K}$: the session key output by stage $s$
- $\pi.stm$: the current position of the protocol's state machine
- $\pi.st[s]$: any additional protocol state values that a previous stage gives as input to stage $s$ (defined as part of the protocol).
- $\pi.sid[s]$: the identifier of stage $s$ of session $\pi$; this is view the actor has of the session $\pi$ in stage $s$, as defined in Fig. 6.
- $\pi.type[s]$: 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 $\pi$ 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 identify sessions and partners.

#### 4.1.1. *Instantiating Signal in Terms of Definition 1*

The KeyGen and MedTermKeyGen algorithms correspond to the first two lines of Fig. 6a.

Activate sets each party's intended role, as well as optionally specifies the identifier of their intended peer.

The Run algorithm incorporates the rest of Fig. 6, combined with a state machine deciding which stage to execute based on the current state of the session (stored in the variable $\pi.stm$) and the input $m$. From the perspective of Run's state machine, the input $m$ may be either a distinguished control message (which the user or adversary uses to unilaterally direct the instance to enter a new stage), or a protocol message, in which case we assume that metadata present in the incoming real-world protocol message uniquely directs the state machine how to advance.

The initiator and responder each learn and set *peeripk* and *peerprepk* during the execution of Run in Fig. 6b; the responder also learns and sets *peerid* during execution. In remaining calls to the Run algorithm, additional variables are set for each stage. Each stage's state type (*type*) is set when the stage begins, and the stage's status is set to `active` at the same time. The stage's session identifier (*sid*) and state (*st*) are set within the stage as soon as the values indicated in Table 2 have been derived. When the stage computes its message key *mk*, it saves that as the session key ($k$) and sets the stage's status to `accept`.

### 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 $\Pi$ 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

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

where the security experiment $\mathsf{Exp}^{\text{ms-ind}}_{\Pi,\mathsf{n_P},\mathsf{n_M},\mathsf{n_S},\mathsf{n_s}}(\mathcal{A})$ is as defined in Fig. 8. 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 $\mathsf{Exp}^{\text{ms-ind}}$ includes the following global variables:

- $\mathsf{b}$: 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 $\mathsf{Test}$ query has happened

Furthermore, $\mathsf{Exp}^{\text{ms-ind}}$ extends the per-session state $\pi_u^i$ with the following experiment variables:

- $\pi_u^i.\mathsf{rand}[s] \in \{0,1\}^\lambda$: random coins for $\pi_u^i$'s $s$th stage
- $\pi_u^i.\mathsf{preid} \in \{1, \ldots, \mathsf{n_M}\}$: the index of the owner's medium-term key to use
- $\pi_u^i.\mathsf{peerpreid} \in \{1, \ldots, \mathsf{n_M}\}$: the index of the alleged peer's medium-term key
- $\pi_u^i.\mathsf{rev\_session}[s] \in \{\text{true}, \text{false}\}$: whether $\mathsf{RevSessKey}(u, i, s)$ was called or not; default false
- $\pi_u^i.\mathsf{rev\_random}[s] \in \{\text{true}, \text{false}\}$: whether $\mathsf{RevRand}(u, i, s)$ was called or not; default false
- $\pi_u^i.\mathsf{rev\_state}[s] \in \{\text{true}, \text{false}\}$: 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 can be 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. (In Signal, the initiator will be activated with the identifier of their intended peer, but the responder will only learn the identity of their intended peer during protocol execution.) Certain aspects of the experiment require the administrative index of the corresponding key, and thus, we assume that $\pi_u^i.peerid$ is set to the corresponding identity upon $\pi_u^i.peeripk$ being set, and similarly for the semi-static key index $\pi_u^i.\mathsf{peerpreid}$ upon $\pi_u^i.peerprepk$ 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. Note that our session identifiers include the public identity keys of both parties, which means that our security notion enforces agreement on the identity keys. However, we note that Signal's key derivation or associated data of encrypted messages do not including any application-level identities, such as a phone number. Our session identifiers mirror this choice. This means that the specific type of Unknown Key Share (UKS) attack on TextSecure [36] is not considered an attack in our model: the initiator Alice's session with Eve (with public identity key $ipk^E = ipk^B$) will have the same session identifier as the responder Bob's session with Alice.

**Table 2.** Values that Signal assigns to the per-stage variables in the security model .

| Stage $s$ | Session id $\pi.sid[s]$ | Input state $\pi.st[s]$ | Session key $\pi.k[s]$ | Output state(s) | Randomness $\pi.\mathsf{rand}[s]$ |
|---|---|---|---|---|---|
| 0, *type*[0] = `triple` | (`triple` : $ipk^i, ipk^r, prepk^r,$ $epk^i, rchpk_0^i$) | $\perp$ | $mk^{ir}_{0,0}$ | $\pi.st$[**sym-ir**:0, 1] $\leftarrow ck^{ir}_{0,1}$<br>$\pi.st$[**asym-ri**:1] $\leftarrow rk_1, [rchpk_0^i]^\dagger$ | $[ek^i, rchk_0^i]^\dagger$ |
| 0, *type*[0] = `triple+DHE` | (`triple+DHE` : $ipk^i, ipk^r, prepk^r,$ $epk^i, eprepk^r, rchpk_0^i$) | $\perp$ | $mk^{ir}_{0,0}$ | as for *type*[0] = `triple` | $[ek^i, rchk_0^i]^\dagger$<br>$[eprek^r]^\ddagger$ |
| **asym-ri**:1 | $sid[0] \parallel$ (`asym-ri` : $rchpk_0^i, rchpk_1^r$) | $rk_1$ | $mk^{ri}_{1,0}$ | $\pi.st$[**sym-ri**:1, 1] $\leftarrow ck^{ri}_{1,1}$<br>$\pi.st$[**asym-ir**:1] $\leftarrow tmp, [rchk_1^r]^\ddagger$ | $[rchk_1^r]^\ddagger$ |
| **asym-ri**:$x$, $x \geq 2$ | $sid$[**asym-ir**:$x-1$] $\parallel$ (`asym-ri` : $rchpk_x^r$) | $rk_x$ | $mk^{ri}_{x,0}$ | $\pi.st$[**sym-ri**:$x$, 1] $\leftarrow ck^{ri}_{x,1}$<br>$\pi.st$[**asym-ir**:$x$] $\leftarrow tmp, [rchk_x^r]^\ddagger$ | $[rchk_x^r]^\ddagger$ |
| **asym-ir**:$x$, $x \geq 1$ | $sid$[**asym-ri**:$x$] $\parallel$ (`asym-ir` : $rchpk_x^i$) | $tmp$ | $mk^{ir}_{x,0}$ | $\pi.st$[**sym-ri**:$x$, 1] $\leftarrow ck^{ri}_{x,1}$<br>$\pi.st$[**asym-ri**:$x+1$] $\leftarrow rk_{x+1}, [rchk_x^i]^\dagger$ | $[rchk_x^i]^\dagger$ |
| **sym-ri**:$x$, $y$, $x \geq 1$, $y \geq 1$ | $sid$[**asym-ri**:$x$] $\parallel$ (`sym-ri` : $y$) | $ck^{ri}_{x,y}$ | $mk^{ri}_{x,y}$ | $\pi.st$[**sym-ri**:$x\,y+1$] $\leftarrow ck^{ri}_{x,y+1}$ | $\perp$ |
| **sym-ir**:0, $y$, $y \geq 1$ | $sid[0] \parallel$ (`sym-ir` : $y$) | $ck^{ir}_{0,y}$ | $mk^{ir}_{0,y}$ | $\pi.st$[**sym-ir**:0, $y+1$] $\leftarrow ck^{ir}_{0,y+1}$ | $\perp$ |
| **sym-ir**:$x$, $y$, $x \geq 1$, $y \geq 1$ | $sid$[**asym-ir**:$x$] $\parallel$ (`sym-ir` : $y$) | $ck^{ir}_{x,y}$ | $mk^{ir}_{x,y}$ | $\pi.st$[**sym-ir**:$x$, $y+1$] $\leftarrow ck^{ir}_{x,y+1}$ | $\perp$ |

$[\ldots]^\dagger$ denotes something that applies only to the initiator; $[\ldots]^\ddagger$ applies only to the responder. See Fig. 6 for definition of all values

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

1: $\mathsf{b} \xleftarrow{\$} \{0,1\}$
2: $\mathsf{tested} \leftarrow \bot$
3: *// generate long-term and semi-static keys*
4: **for** $u = 1$ **to** $\mathsf{n_P}$ **do**
5: $\quad (ipk^u, ik^u) \xleftarrow{\$} \mathrm{KeyGen}()$
6: $\quad$ **for** $\mathsf{preid} = 1$ **to** $\mathsf{n_M}$ **do**
7: $\quad\quad (prepk^u_{\mathsf{preid}}, prek^u_{\mathsf{preid}}) \xleftarrow{\$} \mathrm{MedTermKeyGen}(ik^u)$
8: $pubinfo \leftarrow (ipk^1, \ldots, ipk^{\mathsf{np}}, prepk^1_1, \ldots, prepk^{\mathsf{np}}_{\mathsf{nM}})$
9: $\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathsf{Send},\mathsf{Rev*},\mathsf{Test}}(pubinfo)$
10: **if** $(\mathsf{tested} = \bot) \vee \neg\mathsf{fresh}(\mathsf{tested})$ **then**
11: $\quad r \xleftarrow{\$} \{0,1\}$
12: $\quad$ **return** $r$
13: **if** $\mathsf{b} = \mathsf{b}'$ **then**
14: $\quad$ **return** $1$ *// the adversary wins*
15: **else**
16: $\quad$ **return** $0$ *// the adversary loses*

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

1: **if** $\pi^i_u = \bot$ **then**
2: $\quad$ *// start new session and record intended peer*
3: $\quad$ parse $m$ as $(peerid, \mathsf{preid}, role)$
4: $\quad \pi^i_u.\overrightarrow{\mathsf{rand}} \xleftarrow{\$} \{0,1\}^{\mathsf{n_s}\times\lambda}$
5: $\quad (\pi^i_u, m') \leftarrow \mathrm{Activate}(ik^u, prek^u_{\mathsf{preid}}, role, peerid; \pi^i_u.\mathsf{rand}[0])$
6: $\quad$ **return** $m'$
7: **else**
8: $\quad s \leftarrow \pi^i_u.stage$
9: $\quad (\pi^i_u, m') \leftarrow \mathrm{Run}(ik^u, prek^u_{\pi^i_u.\mathsf{preid}}, \pi^i_u, m; \pi^i_u.\mathsf{rand}[s])$
10: $\quad$ **return** $m'$

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

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

$\mathsf{RevLongTermKey}(u)$:

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

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

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

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

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

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

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

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

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

**Fig. 8.** Security experiment for adversary $\mathcal{A}$ against multi-stage key indistinguishability security of protocol $\Pi$ .

### 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 Fig. 3. 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 Fig. 3 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 Diffie–Hellman (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.

The freshness predicate $\mathsf{fresh}$ 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. In turn, each of these sub-predicates is themselves based on a variety of underlying predicates (for example, $\mathsf{clean}_{\texttt{EL}}$, $\mathsf{clean}_{\texttt{EM}}$, and $\mathsf{clean}_{\texttt{EE}}$). These sub-predicates are introduced in Sects. 4.3.1, 4.3.2, and 4.3.3.

**Definition 4.** (*Validity and freshness*) Let $s$ be a stage in the $i$th session at agent $u$, and let $\tau = \pi_u^i.type[s]$ be its type (e.g. `triple`, `triple+DHE`, ...) as specified in Table 2. It is $\mathsf{valid}$ 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 $\mathsf{fresh}$ if it additionally satisfies cleanness:

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

$\mathsf{fresh}$ 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` key exchange is derived from the concatenation of three Diffie–Hellman (DH) shared secrets, combined with one more DH shared secret; thus, it will be secret as long as at least one of the four input secrets is. The cleanness predicate in a stage of this type is thus the disjunction of several predicates, each encoding the secrecy of one Diffie–Hellman (DH) pair: a long-term/medium-term pair, an ephemeral/long-term pair, and an ephemeral/medium-term pair.[7]

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

[7] In our model, there are two ephemeral/medium-term pairs: $(prepk^B)^{ek^A}$ and $(prepk^B)^{rchk_0^A}$. Our security model treats $ek^A$ and $rchk_0^A$ as being revealed by the same query, so one predicate covers both terms.

**Definition 5.** ($\mathsf{clean}_{\mathtt{triple}}$) Let

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

For a generic sub-clause $\mathsf{clean}_{\mathtt{XY}}$, 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 Fig. 3.

For the session key derived from a `triple+DHE` key exchange, there is an additional Diffie–Hellman (DH) shared secret from an ephemeral/ephemeral pair:

**Definition 6.** ($\mathsf{clean}_{\mathtt{triple+DHE}}$) Let

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

We now define each of the sub-clause of Definitions 5 and 6. Each of these sub-clause is defined differently depending on whether it is evaluated for an initiator or responder session.

For the long-term/medium-term sub-clause, we simply need to check that neither the initiator's long-term key nor the responder's medium term-key has been revealed:

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

For sub-clauses involving ephemeral keys, we have to consider more subtle aspects. For responder sessions, the difficult part is that the ephemeral key is now the peer's, not the session owner'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); this will be what the clause $\mathsf{clean}_{\mathtt{peerE}}$ does.

For the ephemeral/long-term and ephemeral/medium-term sub-clauses, we check that the initiator's ephemeral key and the responder's long-term or medium-term key has not been revealed, using $\mathsf{clean}_{\mathtt{peerE}}$ to address the issue in the previous paragraph:

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

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

When the session $\pi_u^i$ is the responder, if the medium-term key of the responder is corrupted, then we do not permit an attack impersonating Alice to Bob: since the only randomness in the initial key exchange phase is from the initiator (and there is no static-static Diffie–Hellman (DH) secret), such an attack will succeed, and hence an adversary that corrupts the medium-term key of the responder can easily impersonate Alice to Bob.

For the ephemeral/ephemeral sub-clauses, we check that neither party's ephemeral key has been revealed, again using $\mathsf{clean}_{\texttt{peerE}}$ to ensure that the peer session actually exists:

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

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.

The $\mathsf{clean}_{\texttt{peerE}}(u,i,s)$ sub-clause that we relied on to deal with the issue of the ephemeral key at the peer is given below. However, there is a special case: a session $\pi_u^i$ with $\pi_u^i.role = \texttt{init}$, in stage $s = 0$ with $type[0] = \texttt{triple+DHE}$. Specifically, we need to confirm an honest responder exists that has generated the ephemeral prekey $eprepk^r$ that was received by $\pi_u^i$. The reason why we treat this separately is that all other ephemeral–ephemeral (i.e. ratchet) keys are separated into distinct stages, which is not the case for stage $s = 0$ with $type[0] = \texttt{triple+DHE}$. In the other cases, the $\mathsf{clean}_{\texttt{peerE}}(u,i,s)$ predicate determines whether there exists a partner session that has generated the received ephemeral key $rchpk_{x-1}^i$ from the previous stage, and whether that $rchk_{x-1}^i$ has been compromised via a RevRand query. This is trickier to determine for the initial stage, since the session identifier *sid* is set when the session keys are derived, and thus we cannot use *sid* to determine if an honest session $\pi^j_{\pi_u^i.peerid}.role = \texttt{resp}$ has generated the ephemeral prekey without also requiring that the responder has received the first message from the initiator. Instead, we use $\mathsf{rand}[s]$ to determine whether there exists an honest responder $\pi^j_{\pi_u^i.peerid}$ that has generated the ephemeral prekey $eprepk^j$ that the initiator has received. For readability reasons, in what follows we denote the public key *eprepk* associated with the ephemeral randomness $eprek^j = \pi^j_{\pi_u^i.peerid}.\mathsf{rand}[s]$ as $eprepk^j$ (i.e. $eprepk^j = g^{eprek^j}$). We also use $x \subset y$ to denote "$x$ is a substring of $y$".

$$\mathsf{clean}_{\texttt{peerE}}(u,i,s) = \begin{cases} \exists\, j : eprepk^j \subset \pi_u^i.sid[s] & \pi_u^i.role = \texttt{init}, \text{ and} \\ \quad \wedge\Big(\forall\, j : eprepk^j \subset \pi_u^i.sid[s] \implies & type[0] = \texttt{triple+DHE} \\ \qquad \neg\pi^j_{\pi_u^i.peerid}.\mathsf{rev\_random}[s]\Big) & \\ \exists\, j : \pi_u^i.sid[s] = \pi^j_{\pi_u^i.peerid}.sid[s] & \\ \quad \wedge\Big(\forall\, j : \pi_u^i.sid[s] = \pi^j_{\pi_u^i.peerid}.sid[s] \implies & \text{otherwise} \\ \qquad \neg\pi^j_{\pi_u^i.peerid}.\mathsf{rev\_random}[s]\Big) & \end{cases}$$

*Excluded attacks.* Recall that a vertex cover of Fig. 3 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 $B$ ($prek^B$, $ik^B$), 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 [45]; it is ruled out in our model because $\mathsf{clean}_{\texttt{triple}}$ 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 Diffie–Hellman (DH) shared secrets into the state, whereas symmetric ratcheting solely applies a KDF to existing state. During asymmetric ratcheting, there are actually two substages, in which keys with slightly different properties are derived.

In the first substage, of type [**asym-ri**:$x$], the parties apply a KDF to two pieces of keying material: a Diffie–Hellman (DH) shared secret derived from both parties' ratcheting public keys (the initiator's previous ratcheting key and the responder's new ratcheting key), and the root key derived at the end of the previous asymmetric stage. Keys from this substage should be secure if either of the two pieces is unrevealed. However, there are two ways that the root key from the end of the previous asymmetric stage could be considered revealed: either by revealing the *state* passed from one stage to the next (captured by the $\mathsf{clean}_{\mathsf{state}}$ sub-clause following Definition 7) or by the previous stage itself been compromised, which is why we have a conjunction.

**Definition 7.** ($\mathsf{clean}_{\texttt{asym-ri}}$) Let $s_{ri} = [\textbf{asym-ri}{:}x]$ and $s'_{ir} = [\textbf{asym-ir}{:}x-1]$. Define

$$\mathsf{clean}_{\texttt{asym-ri}}(u,i,s_{ri}) = \begin{cases} \mathsf{clean}_{\texttt{EE}}(u,i,[0],[\textbf{asym-ri}{:}1]) \vee \left(\mathsf{clean}_{\mathsf{state}}(u,i,s_{ri}) \wedge \mathsf{clean}_{\pi^i_u.type[0]}(u,i,[0])\right) & x=1 \\ \mathsf{clean}_{\texttt{EE}}(u,i,s_{ri},s'_{ir}) \vee \left(\mathsf{clean}_{\mathsf{state}}(u,i,s_{ri}) \wedge \mathsf{clean}_{\texttt{asym-ir}}(u,i,s'_{ir})\right) & x \geq 2 \end{cases}$$

Note that to capture that the state passed from one stage to the next is not revealed, we use the following clause, which considers revealing the state at the session or at any of its peers, since those peers would have the same state:

$$\mathsf{clean}_{\mathsf{state}}(u,i,s) = \neg\pi^i_u.\mathsf{rev\_state}[s] \wedge \left(\forall\, j : \pi^i_u.sid[s] = \pi^{j}_{\pi^i_u.peerid}.sid[s] \implies \neg\pi^{j}_{\pi^i_u.peerid}.\mathsf{rev\_state}[s]\right)$$

In the second asymmetric ratchet substage, of type [**asym-ir**:$x$], the parties effectively apply a KDF to two sources of keying material: the temporary value $tmp$, computed during the previous substage, and a Diffie–Hellman (DH) shared secret derived from one party's previous ratcheting public key and the other's new ratcheting public key. Keys from this substage should be secure if at least one of the two sources is unrevealed.

**Definition 8.** ($\mathsf{clean}_{\texttt{asym-ir}}$) Let $s_{ir} = [\textbf{asym-ir}{:}x]$ and $s_{ri} = [\textbf{asym-ri}{:}x]$. Define

$$\mathsf{clean}_{\texttt{asym-ir}}(u,i,s_{ir}) = \mathsf{clean}_{\texttt{EE}}(u,i,s_{ir},s_{ri}) \vee \left(\mathsf{clean}_{\mathsf{state}}(u,i,s_{ir}) \wedge \mathsf{clean}_{\texttt{asym-ri}}(u,i,s_{ri})\right)$$

Initially, it may not be clear why we define $\mathsf{clean}_{\texttt{asym-ir}}$ and $\mathsf{clean}_{\texttt{asym-ri}}$ separately—indeed, the definitions appear almost identical. This separation is due to how we define stages, more specifically when per-stage randomness is *generated*, and when the randomness is *used*. Informally, when the initiator begins an asymmetric stage $[\textbf{asym-ir}{:}x]$, the initiator generates its per-stage randomness (i.e. $rchk_x^i$) and uses the responder's previous ratchet key, which was generated in the stage $[\textbf{asym-ri}{:}x]$ (i.e. $rchk_x^r$) with the same stage index $x$. However, when the responder begins an asymmetric stage, the stage index is incremented such that the new stage is $[\textbf{asym-ri}{:}x+1]$, and the responder uses the initiator's previous ratchet key, generated during stage $[\textbf{asym-ir}{:}x]$. This difference between which stage's ephemeral randomness is considered for the cleanness predicate is why we separate the asymmetric cleanness definitions.

Together, the $\mathsf{clean}_{\texttt{asym-ir}}$ and $\mathsf{clean}_{\texttt{asym-ri}}$ clauses capture the "future secrecy" or "post-compromise security" 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 (the $\mathsf{clean}_{\texttt{EE}}$ sub-clause *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.

### 4.3.3. *Symmetric Stages* [***sym-ir***:$x$, $y$] *and* [***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.** ($\mathsf{clean}_{\texttt{sym}}$) Writing $s = [\textbf{sym-ir}{:}x, y]$,

$$\mathsf{clean}_{\texttt{sym-ir}}(u,i,s) = \mathsf{clean}_{\mathsf{state}}(u,i,s) \wedge \begin{cases} \mathsf{clean}_{\pi_u^i.type[0]}(u,i,[0]) & x=0, y=1 \\ \mathsf{clean}_{\texttt{asym-ir}}(u,i,[\textbf{asym-ir}{:}x]) & x \geq 1, y=1 \\ \mathsf{clean}_{\texttt{sym-ir}}(u,i,[\textbf{sym-ir}{:}x, y-1]) & x \geq 0, y \geq 2 \end{cases}$$

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

$$\mathsf{clean}_{\texttt{sym-ri}}(u,i,s) = \mathsf{clean}_{\mathsf{state}}(u,i,s) \wedge \begin{cases} \mathsf{clean}_{\texttt{asym-ri}}(u,i,[\textbf{asym-ri}{:}x]) & x \geq 1, y = 1 \\ \mathsf{clean}_{\texttt{sym-ri}}(u,i,[\textbf{sym-ri}{:}x, y-1]) & x \geq 1, y \geq 2 \end{cases}$$

We may write $\mathsf{clean}_{\texttt{sym}}$ 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 message 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 Sect. 4, under standard assumptions on the cryptographic building blocks.

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

We have made a few minor reorganizations in Fig. 6 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 initializes 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 Fig. 6.

KeyGen and MedTermKeyGen consist of uniform random sampling from the group. Activate depends on the invoked role. Our prekey reorganization described above means that the roles of initiator and responder are technically reversed: although intuitively Alice initiates a session in our presentation, in fact Bob sends the first message, namely his prekeys (first right-to-left flow of Fig. 6b. 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 Fig. 6b and outputs a message containing $epk^A$; for Bob, it operates as in the right side of Fig. 6b 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 Fig. 6c).

(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 Fig. 6d. Advance both chains by one step, and output the message keys as the session key for the two asymmetric substages 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 Fig. 6c) 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.

A high-level overview of the proof with its main game sequences and case distinctions is given in Appendix in Fig. 9. In addition, the full proof is given in "Appendix B.3".

Signal has many different types of stages, 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 Diffie–Hellman (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, we emphasize, 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 analysing stage [0], and if the stage type is `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 the standard multi-stage indistinguishability game 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 Diffie–Hellman (DH) public values 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 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 three 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 four 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 $\pi_u^i.type[0] = \texttt{triple}$, we see by Definition 5 that the following condition must be satisfied:

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

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

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

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 7: 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 Diffie–Hellman (DH) shares in the ephemeral shares used in the stage (which in particular were generated

in stage $s = [0]$). That is, we consider the case that $\mathsf{Test}(u, i, s = [\textbf{asym-ri}{:}1])$ where $\pi_u^i.type[s] = \texttt{asym-ri}$ and apply Definition 7 to conclude that

$$\left(\mathsf{clean}_{\pi_u^i.type[0]}(u, i, [0]) \quad \wedge \quad \mathsf{clean}_{\mathsf{state}}(u, i, [\textbf{asym-ir}{:}1])\right) \quad \vee \quad \mathsf{clean}_{\mathrm{EE}}(u, i, 0, 0)$$

In a similar fashion to the argument for the initial handshake, we replace certain Diffie–Hellman (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 $x^{\text{th}}$ asymmetric responder-to-initiator stage $s = [\textbf{asym-ri}{:}x]$. That is, we consider the case that $\mathsf{Test}(u, i, s = [\textbf{asym-ri}{:}x])$, where $x \geq 2$ and $\pi_u^i.type[s] = \texttt{asym-ri}$. By Definition 7, we conclude:

$$\left(\mathsf{clean}_{\texttt{asym-ri}}(u, i, [\textbf{asym-ri}{:}x-1]) \wedge \mathsf{clean}_{\mathsf{state}}(u, i, [\textbf{asym-ir}{:}x])\right) \vee \mathsf{clean}_{\mathrm{EE}}(u, i, x-1, x-1)$$

and a similar argument holds.

We must also consider the case that the $\mathsf{Test}$ 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

*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 Diffie–Hellman (DH) agreement and for signing the medium-term prekeys.[8] [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 prekey.

*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 Sect. 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,[9] 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

[8] This is done in practice by reinterpreting the Curve25519 point as an Ed25519 key and computing an EdDSA signature.

[9] 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.

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. As a result, our analysis has a significantly large factor in the non-tightness of our security reduction. Specifically, we lose *at minimum* a factor of $(n_P^2 \cdot n_S)$ to our reduction to the Gap Diffie–Hellman assumption. Unfortunately, attempting to instantiate the Signal protocol with parameter sizes that our analysis suggests would be secure would *not* be compatible with parameter sizes that Signal implementations currently use. Indeed, implementing Signal with such parameter sizes would incur significantly more computational cost, a difficult proposition for mobile devices. One might argue about the practical relevance of such an analysis; however, the current proof does provide quantitative statements about the security of the Signal protocol, as well as a qualitative indicator of the general soundness of the design; moreover, the structure of the proof itself may be useful in future research as a starting point to increase the tightness of the security reduction.

### *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 resend 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, which we prove that Signal satisfies under standard cryptographic assumptions. Practically speaking, they imply secrecy and authentication of the message keys which Signal derives, even under a variety of adversarial compromise scenarios such as forward security (and thus "future secrecy"). If used correctly, Signal could achieve a form of post-compromise security, which has substantial advantages over forward secrecy as described in [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 [51] or including a static–static Diffie–Hellman (DH) shared secret in the key derivation.

We have described some of the limitations of our approach in Sect. 6. Furthermore, the complexity and tendency to add "extra features" make 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.

Our security model itself is quite general and is well-suited for the analysis of other messaging protocols. In particular, we see our novel extensions to multi-stage key-exchange models that capture medium-term keys to be applicable to communication protocols outside of Signal. This is in comparison with our freshness definition, which is tailored specifically to the structure of the Signal protocol. Our presentation of the Signal protocol itself as well as the approach taken in our analysis may be of independent interest and help guide future investigation of the Signal protocol, and other cryptographic protocols of similar complexity.

**Acknowledgements**

The authors acknowledge helpful discussions with Marc Fischlin and Felix Günther and valuable comments from Chris Brzuska and Trevor Perrin. Thanks to Xavier Bultel for carefully reading through the proof. Thanks to Mang Zhao for identifying a missing DH operation in the session setup phase. K. C-G. and L.G. were supported by the Oxford CDT in Cyber Security. B.D was supported by EPSRC Grant EP/L018543/1. D.S. was supported by Australian Research Council (ARC) Discovery Project Grant DP130104304, Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2016-05146, and NSERC Discovery Accelerator Supplement Grant RGPIN-2016-05146.

## 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 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 Diffie–Hellman (DH) keys with random values, we must ensure that the resulting game is similar to its original. For protocols using only ephemeral Diffie–Hellman (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 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* 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-Diffie–Hellman (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 Diffie–Hellman (DH) secret. If so, the simulation can terminate and the challenger uses this value in the Gap-Diffie–Hellman (DH) game. This is the approach we take.

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

There is a further complication in the case of Signal. In most normal Diffie–Hellman (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 Diffie–Hellman (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 PRF-ODH game parameterized 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-Diffie–Hellman (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 PRF-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 Diffie–Hellman (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 $\mathcal{O}_{\mathrm{DDH}}$ 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

$$\epsilon_{\mathrm{DDH}}(\mathcal{D}) := \left| \Pr\left[\mathcal{D}(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}, g^{\alpha\beta}) = 1 : \alpha, \beta \xleftarrow{\$} \mathbb{Z}_q\right] - \Pr\left[\mathcal{D}(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}, g^{\gamma}) = 1 : \alpha, \beta, \gamma \xleftarrow{\$} \mathbb{Z}_q\right] \right|$$

$$\epsilon_{\mathrm{CDH}}(\mathcal{D}) := \Pr\left[\mathcal{D}(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}) = g^{\alpha\beta} : \alpha, \beta \xleftarrow{\$} \mathbb{Z}_q\right]$$

$$\epsilon_{\mathrm{GDH}}(\mathcal{D}) := \Pr\left[\mathcal{D}^{\mathcal{O}_{\mathrm{DDH}}}(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}) = g^{\alpha\beta} : \alpha, \beta \xleftarrow{\$} \mathbb{Z}_q\right]$$

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., $\epsilon_{\mathrm{DDH}}(\mathcal{D})$ 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 $(g^{\alpha}, g^{\beta})$, i.e., $\epsilon_{\mathrm{CDH}}(\mathcal{D})$ 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 $(g^{\alpha}, g^{\beta})$ 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 modifications to the Signal protocol made for our proof in B.1. 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 plaintext 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, parameterized by freshness condition fresh, with non-negligible probability. Suppose for contradiction that such an

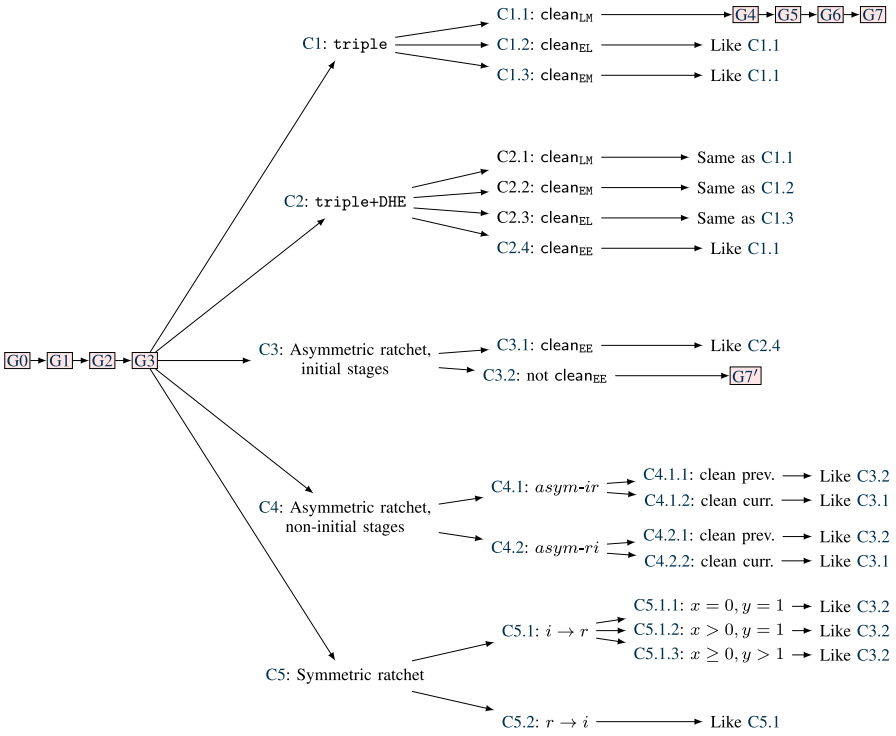



**Fig. 9.** High-level overview of the proof structure. Games are identified by G0, G1, …, and main case distinctions by C1, C2, …; G0 denotes the multi-stage security experiment from Sect. 4.2. In the PDF version of this document, such identifiers can be clicked to jump to the corresponding part of the proof .

adversary $\mathcal{A}$ exists. Whatever the behaviour of the adversary, trivially (by the definition of the security experiment in Fig. 8) 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) $\pi_u^i.\mathit{status}[s] = \texttt{accept}$,
(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 $\pi_u^i.\mathit{peerid}$, 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.

### *High-Level Overview of the Case Distinction*

A high-level overview of the proof with its main game sequences and case distinctions is given in Figure 9.

*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 $\mathsf{clean}_{\texttt{triple}}$ clause—over the different ways the session could be clean—noting that one of $\mathsf{clean}_{\texttt{LM}}(u, i)$, $\mathsf{clean}_{\texttt{EL}}(u, i, 0)$, $\mathsf{clean}_{\texttt{EM}}(u, i, 0)$ 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 $\mathsf{clean}_{\texttt{EE}}$, and for these we obtain a probability bound by replacing the Diffie–Hellman (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 initialize the current sending chain with uniformly random values, since an adversary who could detect this could win against that previous stage.

### 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 Fig. 7) are random oracles and thus need not make any assumptions on HMAC and HKDF specifically.

By $break_i$, we mean the event that the adversary wins game $G_i$. By $\mathsf{Adv}_i$, we mean the advantage of the adversary against game $G_i$; that is,

$$\mathsf{Adv}_i := |2\Pr(break_i) - 1|$$

We aim to show that $\mathsf{Adv}_0$ is a negligible function of the security parameter. 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. See Fig. 9 for a high-level overview of the proof structure and the case distinctions.

*Game Hops for all Cases*

**Game 0**

This game equals the multi-stage security experiment described in Sect. 4.2. As such, the advantage of the adversary against this game is $\mathsf{Adv}_0$.

**Game 1**

In this game, we ensure no collision of honestly generated Diffie–Hellman (DH) public keys. Specifically, the challenger $\mathcal{C}$ maintains a list $L$ of all Diffie–Hellman (DH) private values (for *ik*, *prek*, *ek*, *eprek*, *rchk*) honestly generated during the game. If a Diffie–Hellman (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 $\mathsf{n_M n_P}$ medium-term keys, and a maximum of $\mathsf{n_s}$ ephemeral/ratchet keys per session for a total maximum of $\mathsf{n_S n_s}$ ephemeral/ratchet keys. This means a total maximum of $\mathsf{n_P} + \mathsf{n_P n_M} + \mathsf{n_S n_s}$ 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:

$$\mathsf{Adv}_0 \leq \frac{\binom{\mathsf{n_P}+\mathsf{n_P n_M}+\mathsf{n_S n_s}}{2}}{q} + \mathsf{Adv}_1$$

We now know that from this game onwards each honestly generated Diffie–Hellman (DH) public key is unique. In future game hops, we will replace certain Diffie–Hellman (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 $\pi_u^i$ against which the $\mathsf{Test}(u, i, s)$ query is issued: the challenger guesses a pair of indices $(u^*, i^*) \in$

$[1..\mathsf{n_P}] \times [1..\mathsf{n_S}]$. Let $T$ be the event that the adversary issues a Test query $\mathsf{Test}(u, i, s)$ where $(u, i) \neq (u^*, i^*)$. In this game, we abort if event $T$ occurs; it is a transition based on a large failure event.

$T$ will occur with probability $1/\mathsf{n_S n_P}$, and hence:

$$\mathsf{Adv}_1 = \mathsf{n_S n_P Adv}_2$$

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 $\pi_v^j$ that matches the Test session $\pi_u^i$ but $v \neq v^*$. Note that it might be the case that no such matching $\pi_v^j$ exists, but this game ensures that if such a $\pi_v^j$ 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 $\pi_u^i$ (note $v$ may have multiple sessions that match $\pi_u^i$ 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 Diffie–Hellman (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 (large failure event):

$$\mathsf{Adv}_2 \leq \mathsf{n_P Adv}_3$$

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 $\pi_u^i$ because the adversary may replay $\pi_u^i$'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:

$$\mathsf{Adv}_0 \leq \frac{\binom{\mathsf{n_P}+\mathsf{n_P n_M}+\mathsf{n_S n_s}}{2}}{q} + \mathsf{n_S n_P^2} \cdot \mathsf{Adv}_3$$

At this point, we need to partition our analysis for individual cases, with the ultimate aim of bounding $\mathsf{Adv}_3$ above. Once we have bounded $\mathsf{Adv}_3$, then we have bounded $\mathsf{Adv}_0$ and we are done. Since this is $G3$, each different case begins with a hop to some $G4$. Given there are five cases, it is clear that:

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_3^{\mathbf{C1}} + \mathsf{Adv}_3^{\mathbf{C2}} + \mathsf{Adv}_3^{\mathbf{C3}} + \mathsf{Adv}_3^{\mathbf{C4}} + \mathsf{Adv}_3^{\mathbf{C5}}$$

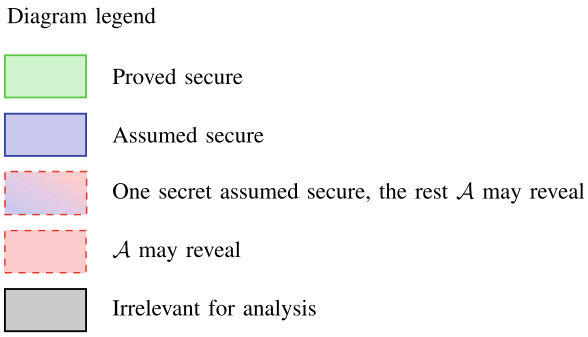



**Fig. 10.** Legend for the boxes in the following diagrams. Red boxes indicate secrets that the adversary may gain access to via Reveal queries (or by computing the secrets as a result of the Reveal query), green boxes indicate secrets that are replaced based on the challenge, and blue boxes indicate secrets that the challenger is able to replace with random, thus ensuring security (Color figure online).

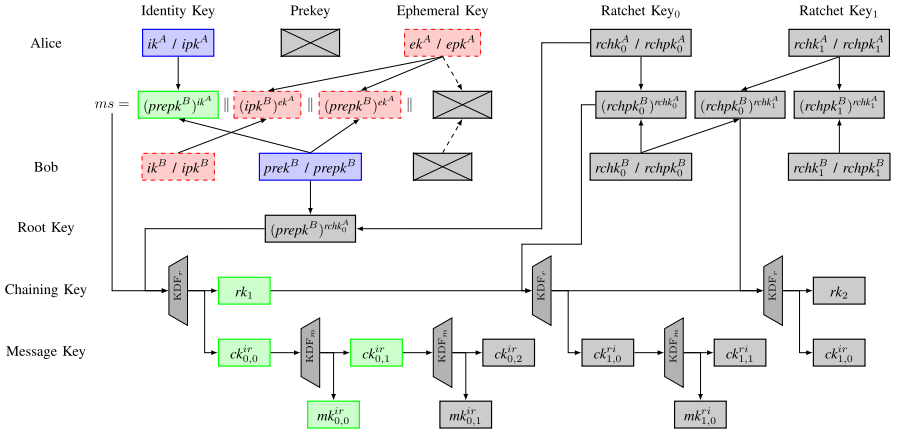



**Fig. 11.** A diagram showing the replacement of secrets in Game 6 of Case 1.1. In particular, denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; denotes secrets that the challenger is able to replace with random, thus ensuring security; and denotes secrets that are not relevant to this case. Diagram legend can be found in Fig. 10 (Color figure online).

### C1: Initial Key Exchange: $type[0] = \texttt{triple}$

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 $\pi_u^i.type[0] = \texttt{triple}$. By construction, the only way for the adversary to win (with non-negligible probability) is if $\mathsf{clean}_{\texttt{triple}}(u, i, [0])$ is true. We partition these scenarios into subcases. Note also that a $\mathsf{RevState}(u, i, [0])$ or $\mathsf{RevState}(v, j, [0])$ (where $\pi_v^j$ is a session matching $\pi_u^i$ 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 $E^{\texttt{triple}}$ 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 $\pi_u^i.type = \texttt{triple}$, and let $E^{\texttt{triple}}_{\mathsf{clean}_{\mathrm{LM}}}$ (resp. $E^{\texttt{triple}}_{\mathsf{clean}_{\mathrm{EL}}}$, $E^{\texttt{triple}}_{\mathsf{clean}_{\mathrm{EM}}}$) be the sub-case in which additionally $\mathsf{clean}_{\mathrm{LM}}(u, i)$ (resp. $\mathsf{clean}_{\mathrm{EL}}(u, i, [0])$, $\mathsf{clean}_{\mathrm{EM}}(u, i, [0])$ is true. By definition of $\mathsf{clean}_{\texttt{triple}}$:

$$\mathsf{Adv}_3^{\mathbf{C1}} \leq \mathsf{Adv}_3^{\mathbf{C1},\mathsf{clean}_{\mathrm{LM}}(u,i)} + \mathsf{Adv}_3^{\mathbf{C1},\mathsf{clean}_{\mathrm{EL}}(u,i,0)} + \mathsf{Adv}_3^{\mathbf{C1},\mathsf{clean}_{\mathrm{EM}}(u,i,0)}$$

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

In this case, $\mathcal{A}_{\texttt{triple}}$ issued a $\mathsf{Test}(u, i, [0])$ query such that $\mathsf{clean}_{\mathrm{LM}}(u, i)$ 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 $\pi_u^i$ has received $prepk_n^v$ without modification.

Recall that an honest session derives a master secret $ms = g^{ik^u \cdot prek_n^v} \| g^{ek^u \cdot ik^v} \| g^{ek^u \cdot prek_n^v}$, and then assigns $rk_1 \parallel ck_{0,0}^{ir} \leftarrow \mathrm{HKDF}(ms)$.

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 $\mathsf{clean}_{\mathrm{LM}}(u, i)$ 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 ($prek_n^v$) that the Test session will use in the execution of the protocol, and aborts if the guess is wrong. This yields (via large failure event) that

$$\mathsf{Adv}_3 \leq \mathsf{n_M}\mathsf{Adv}_4$$

**Game 5**

In this game, the experiment *does not* abort if $ipk^u$ and $prepk_n^v$ 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

$$\mathsf{Adv}_4 \leq 1/q + \mathsf{Adv}_5$$

**Game 6**

In this game, the experiment aborts if the adversary queries $g^{ik^u \cdot prek_n^v} = \mathrm{CDH}(ipk^u, prepk_n^v)$ as the first component of a call to the HKDF random oracle; denote this event $abort_6$. Thus, by a hop via small failure event,

$$\mathsf{Adv}_5 \leq \mathsf{Adv}(break_6) + \mathsf{Adv}_6$$

We now need to bound $\mathsf{Adv}_6$. 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 $(g^\alpha, g^\beta)$ (for $\alpha$ and $\beta$ 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 $g^{xy} = g^z$.

$B_0$ will simulate Game 5, except that it replaces $ipk^u$ with $g^\alpha$ and $prepk_n^v$ with $g^\beta$, refer to Fig. 11. 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 $\alpha$ and $\beta$, 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 $prek_n^v$ 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 $prek_n^v$ 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 $rk_1, ck_{0,0}^{ir}$ rather than deriving them 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 $g^{x_1} \| g^{x_2} \| g^{x_3}$.

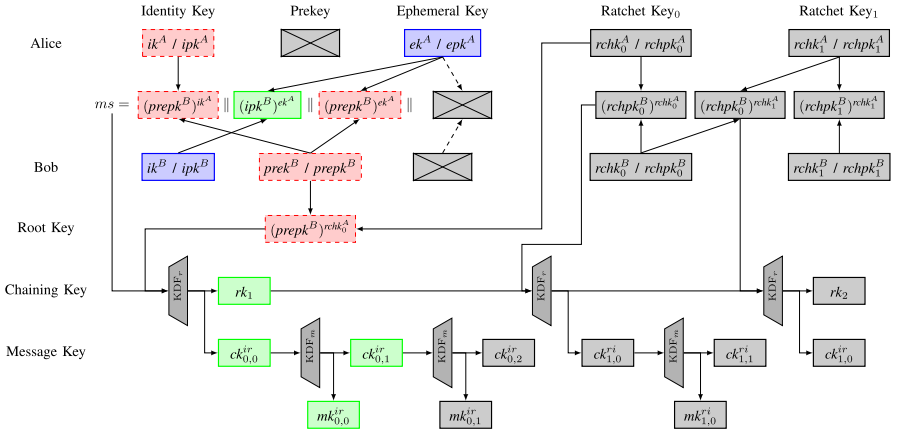



**Fig. 12.** A diagram showing the replacement of secrets in Game 5 of Case 1.2. In particular, denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; denotes secrets that the challenger is able to replace with random, thus ensuring security; and denotes secrets that are not relevant to this case Diagram legend can be found in Fig. 10 (Color figure online).

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 whether 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.

$$\mathsf{Adv}(break_6) = \epsilon_{\mathrm{GDH}}(\mathcal{B}_0)$$

### 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 event $abort_6$ in Game 6, we know that the adversary never queried the random oracle HKDF on the input $ms$ that was used

to compute the session key $rk_1 \parallel ck^{ir}_{0,0}$ of the Test session. Thus, in the random oracle model,

$$\mathsf{Adv}_6 = \mathsf{Adv}_7 = 0$$

since the session key is uniformly random and independent of the hidden bit, and hence the adversary has no advantage in guessing the hidden bit and winning the experiment. We conclude that

$$\mathsf{Adv}_3^{\mathbf{C1},\mathsf{clean}_{\mathrm{LM}}(u,i)} \leq \mathsf{n}_{\mathsf{M}}(1/q + \epsilon_{\mathrm{GDH}}(\mathcal{B}_0) + 0)$$

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

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

**Game 4, 5, 6**

The argument for this case is almost identical to that of the previous subcase, 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 $g^\alpha$, $g^\beta$ 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, refer to Fig. 12.

$$\mathsf{Adv}_3^{\mathbf{C1},\mathsf{clean}_{\mathrm{EL}}(u,i,0)} \leq 1/q + \epsilon_{\mathrm{GDH}}$$

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

**Game 4, 5, 6, 7**

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

Again, this is analogous to before. We begin by guessing the index of the signed prekey of the responder, incurring a factor of $\mathsf{n}_{\mathsf{M}}$. By the definition of the cleanness predicate $\mathsf{clean}_{\mathrm{EM}}(u, i, [0])$, since both the ephemeral key of the initiator *and* the first ratchet key of the initiator are generated during the initial key exchange and used to derive the first message key, we might use either the ratchet key or the ephemeral key of the initiator as the basis for our game hop. We choose to embed the Gap-DH challenge values $g^\alpha$, $g^\beta$

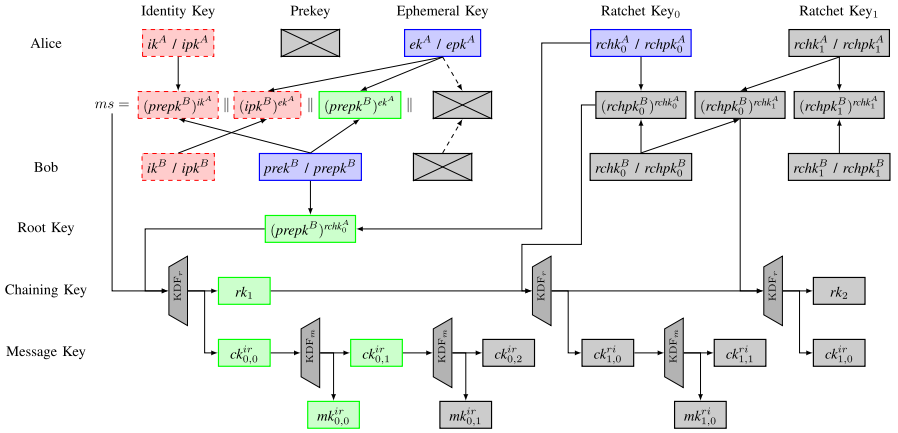



**Fig. 13.** A diagram showing the replacement of secrets in Game 6 of Case 1.3. In particular, denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; denotes secrets that the challenger is able to replace with random, thus ensuring security; and denotes secrets that are not relevant to this case Diagram legend can be found in Fig. 10 (Color figure online).

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, refer to Fig. 13. Thus,

$$\mathsf{Adv}_3^{\mathbf{C1},\mathsf{clean}_{\mathrm{EM}}(u,i,0)} \leq \mathsf{n}_{\mathsf{M}} \cdot (1/q + \epsilon_{\mathrm{GDH}})$$

### C2: Initial key exchange: *type*[0] = `triple+DHE`

Recall that the initial key exchange can also have type `triple+DHE`, in which case cleanness requires that

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

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

- $E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\mathrm{LM}}}$ 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 $\mathsf{clean}_{\mathrm{LM}}(u,i)$ is upheld,
- $E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\mathrm{EM}}}$ where $\mathcal{A}$ has issued a Test query $\mathsf{Test}(u,i,[0])$ and $\mathsf{clean}_{\mathrm{EM}}(u,i,[0])$ is upheld,
- $E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\mathrm{EL}}}$ where $\mathcal{A}$ has issued a Test query $\mathsf{Test}(u,i,[0])$ and $\mathsf{clean}_{\mathrm{EL}}(u,i,[0])$ is upheld, and

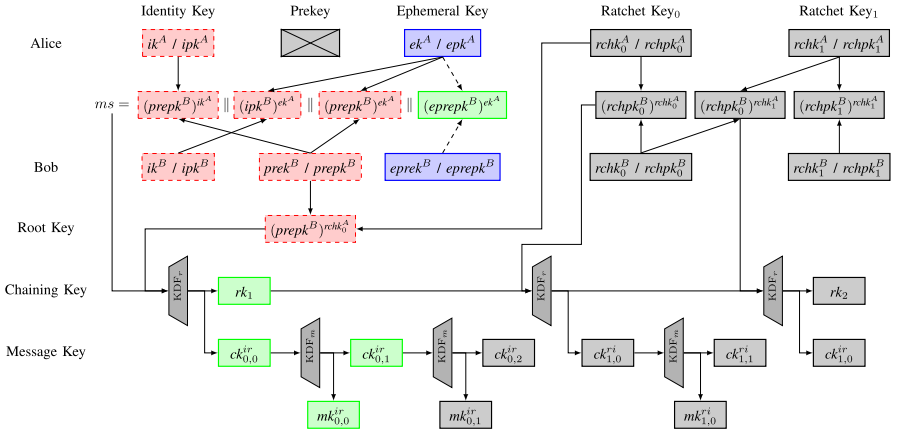



**Fig. 14.** A diagram showing the replacement of secrets in Game 6 of Case 2.4. In particular, [ ] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [ ] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [ ] denotes secrets that the challenger is able to replace with random, thus ensuring security; and [ ] denotes secrets that are not relevant to this case Diagram legend can be found in Fig. 10 (Color figure online).

- $E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\mathrm{EE}}}$ where $\mathcal{A}$ has issued a Test query $\mathsf{Test}(u, i, [0])$ and $\mathsf{clean}_{\mathrm{EE}}(u, i, [0], [0])$ is upheld.

By definition of $\mathsf{clean}_{\texttt{triple+DHE}}$:

$$\mathsf{Adv}_3^{\mathbf{C2}} \leq \mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{LM}}(u,i)} + \mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EL}}(u,i,[0])} + \mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EM}}(u,i,[0])} + \mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EE}}(u,i,[0],[0])}$$

The bounds above 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-Diffie–Hellman (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*$[0] = \texttt{triple+DHE}$ *and* $\mathsf{clean}_{\mathrm{EE}}$

### Game 4, 5, 6, 7

The final ephemeral–ephemeral case $E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\mathrm{EE}}}$ is analogous to previous cases except that in Game 6 ($E^{\texttt{triple+DHE}}_{\mathsf{clean}_{\mathrm{EE}}}$), we need to replace the ephemeral values of both the initiator and the responder, refer to Fig. 14. (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,

$$\mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EE}}(u,i,[0],[0])} \leq \mathsf{n_S} \cdot (1/q + \epsilon_{\mathrm{GDH}})$$

### C3: Asymmetric Ratcheting, Initial Stage

We have now proved security of the initial key exchange, optionally including the ephemeral–ephemeral Diffie–Hellman (DH) computation. We next move on to the asymmetric-ratcheting stages, in which Bob and Alice take turns generating new Diffie–Hellman (DH) ephemeral values 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 $type = \texttt{asym-ri}$. 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,
- $E^{\texttt{asym-ri}}_{\mathsf{clean}_{\mathrm{EE}}(u,i,[0])}$ to be the sub-event of $E^{\texttt{asym-ri}}$ satisfying $\mathsf{clean}_{\mathrm{EE}}(u, i, [0], [0])$, and
- $E^{\texttt{asym-ri}}_{\mathsf{clean}_{\pi_u^i.type[0]}(u,i,[0])}$ to be the sub-event of $E^{\texttt{asym-ri}}$ satisfying

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

It follows from our definition of freshness that

$$\mathsf{Adv}_3^{\mathbf{C3}} \leq \mathsf{Adv}_3^{\mathbf{C3},\mathsf{clean}_{\mathrm{EE}}(u,i,[0],[0])} + \mathsf{Adv}_3^{\mathbf{C3},\mathsf{clean}_{\pi_u^i.type[0]}(u,i,[0])} \tag{1}$$

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

*C3.1: Case* $s = [\textbf{\textit{asym-ri}}{:}1]$, $type[s] = \texttt{asym-ri}$ *and* $\mathsf{clean}_{\mathrm{EE}}(u, i, [0], [0])$

**Game 4, 5, 6, 7**

This case is dealt with similarly to subcase 2.4, with the only substantial differences being that the GDH challenge values are substituted into the ratchet keys $g^{rchk_u^{[0]}}$ and $g^{rchk_v^{[0]}}$ (refer to Fig. 15) and we do not need to guess the index of the ratchet keys. Since the adversary reveals secret ephemeral values for specific stages using the $\mathsf{RevRand}$ query (as opposed to querying specific secret values), the predicate $\mathsf{clean}_{\mathrm{EE}}$ covers secrecy both of the initiator's initial key exchange ephemeral value, and the initiator's

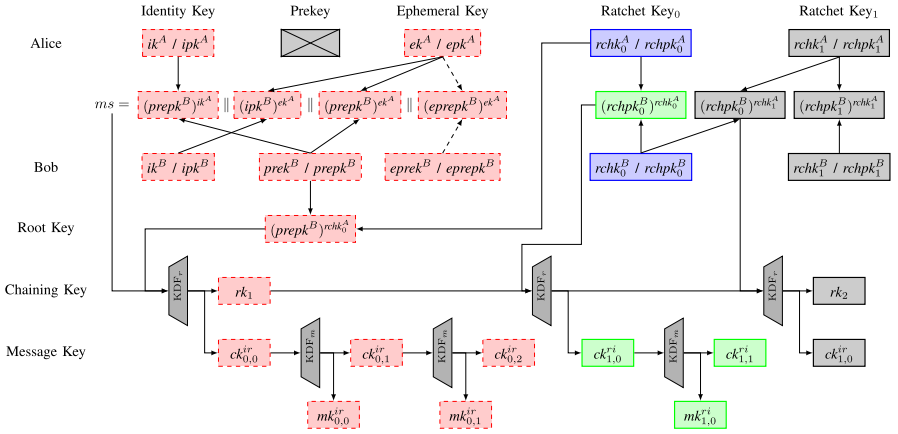



**Fig. 15.** A diagram showing the replacement of secrets in Game 6 of Case 3.1. In particular, [ ] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [ ] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [ ] denotes secrets that the challenger is able to replace with random, thus ensuring security; and [ ] denotes secrets that are not relevant to this case. Diagram legend can be found in Fig. 10 (Color figure online).

first ratchet key, which are generated at the same time. Thus,

$$\mathsf{Adv}_3^{\mathbf{C3},\mathsf{clean}_{\mathrm{EE}}(u,i,[0],[0])} \leq 1/q + \epsilon_{\mathrm{GDH}}$$

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

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 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, refer to Fig. 16.

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 $\mathrm{KDF_r}$ 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 $\mathsf{Adv}_3^{\mathbf{C1}}$ the adversary's advantage in breaking the key-indistinguishability of Case 1, and with $\mathsf{Adv}_3^{\mathbf{C2}}$ we denote the adversary's advantage in breaking the key-indistinguishability of Case 2. Given that only one of Case 1 or Case 2 applies (given how the ini-

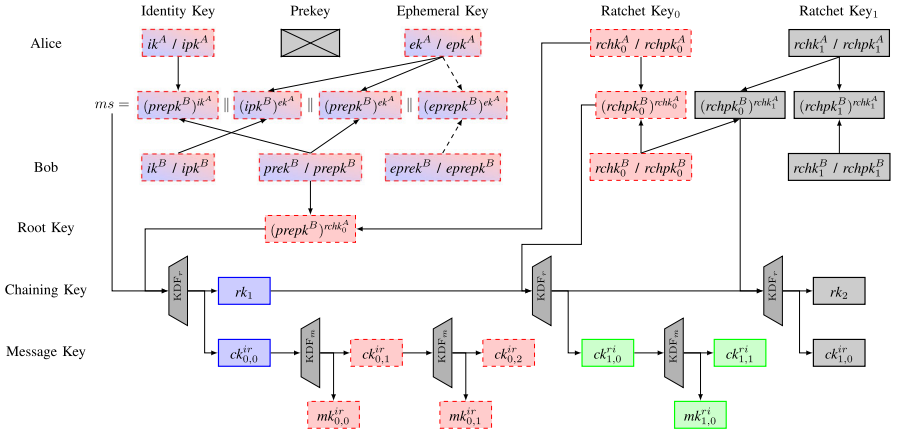



**Fig. 16.** A diagram showing the replacement of secrets in Game 7' of Case 3.2. In particular, [ ] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [ ] denotes secrets that are replaced with the output of a Test query 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 $\mathcal{A}$. Diagram legend can be found in Fig. 10 (Color figure online).

tial key exchange is either of type `triple` or type `triple+DHE`), then the adversary's probability in distinguishing this change is $\max\{\mathsf{Adv}_3^{\mathbf{C1}}, \mathsf{Adv}_3^{\mathbf{C2}}\}$. Note that $\mathsf{Adv}_3^{\mathbf{C1}} \leq \mathsf{Adv}_3^{\mathbf{C1},\mathsf{clean}_{\mathrm{LM}}(u,i)} + \mathsf{Adv}_3^{\mathbf{C1},\mathsf{clean}_{\mathrm{EL}}(u,i,0)} + \mathsf{Adv}_3^{\mathbf{C1},\mathsf{clean}_{\mathrm{EM}}(u,i,0)}$ and $\mathsf{Adv}_3^{\mathbf{C2}} \leq \mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{LM}}(u,i)} + \mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EL}}(u,i,0)} + \mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EM}}(u,i,0)} \mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EE}}(u,i,[0],[0])}$, 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 $ck_{x,0}^{ri}$ and $ck_{x,1}^{ri}$, messaging key $mk_{x,0}^{ri}$, 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. For readability, we define $\mathsf{Adv}_3^{\mathbf{C3},\mathsf{clean}_{\pi_u^i.type[0]}(u,i,[0])} = \mathsf{Adv}_3^{\mathbf{C3.2}}$. Thus,

$$\mathsf{Adv}_3^{\mathbf{C3.2}} \leq \max\{\mathsf{Adv}_3^{\mathbf{C1}} + \mathsf{Adv}_3^{\mathbf{C2}}\}$$

### 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:

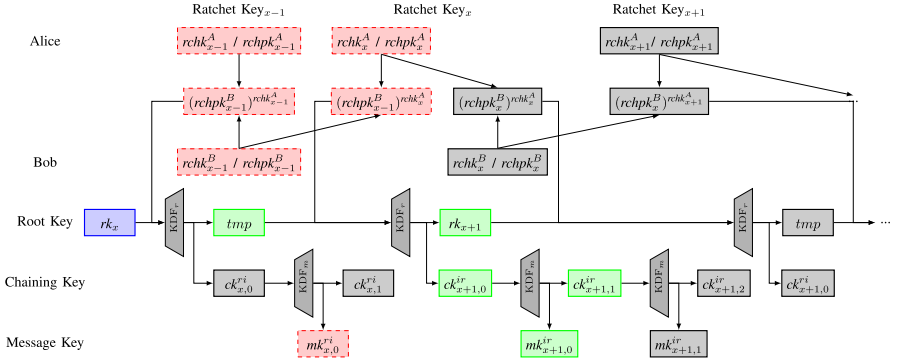



**Fig. 17.** A diagram showing the replacement of secrets in Game 7' of Case 4.1.1. In particular, [ ] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [ ] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [ ] denotes secrets that the challenger is able to replace with random, thus ensuring security; and [ ] denotes secrets that are not relevant to this case. Diagram legend can be found in Fig. 10 (Color figure online).

- 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 (Case 4.1: `asym-ir` vs. Case 4.2: `asym-ri`) and then deal with these cases in turn. Thus, $\mathsf{Adv}_3^{\mathbf{C4}} \leq \mathsf{Adv}_3^{\mathbf{C4.1}} + \mathsf{Adv}_3^{\mathbf{C4.2}}$.

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

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 $E^{\texttt{asym-ir}}_{\text{clean-cur}}$: $\mathsf{clean}_{\mathrm{EE}}(u, i, x-1, x-1)$

Thus,

$$\mathsf{Adv}_3^{\mathbf{C4.1}} \leq \mathsf{Adv}_3^{\mathbf{C4.1},\mathsf{clean}_{\mathsf{prev}}} + \mathsf{Adv}_3^{\mathbf{C4.1},\mathsf{clean}_{\mathrm{EE}}(u,i,x-1,x-1)}$$

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

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, refer to Fig. 17. Treating the KDF as a random oracle, this stage's message key, the next root key $rk_{x+1}$ and the symmetric chaining

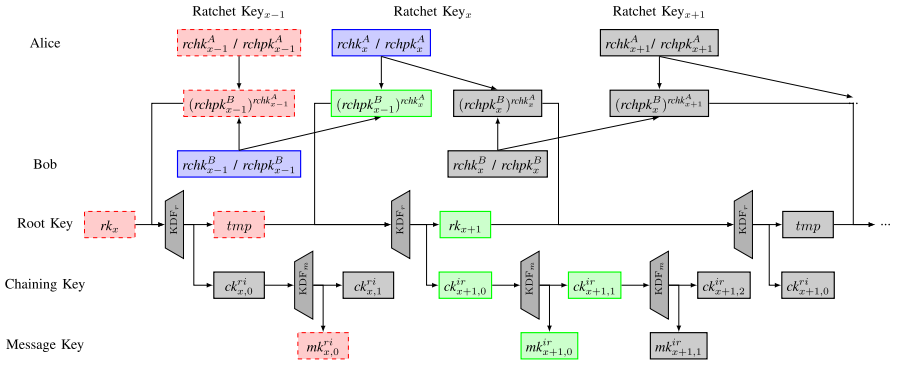



**Fig. 18.** A diagram showing the replacement of secrets in Game 7' of Case 4.1.2. In particular, [ ] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [ ] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [ ] denotes secrets that the challenger is able to replace with random, thus ensuring security; and [ ] denotes secrets that are not relevant to this case. Diagram legend can be found in Fig. 10 (Color figure online).

keys $ck^{ir}_{x,0}$ and $ck^{ir}_{x,1}$ are then indistinguishable from random. For readability, we denote $\mathsf{Adv}_3^{\mathbf{C4.1},\mathsf{clean}_{\mathsf{prev}}} = \mathsf{Adv}_3^{\mathbf{C4.1.1}}$, $\mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{LM}}(u,i)} = \mathsf{Adv}_3^{\mathbf{C2.1}}$, $\mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EL}}(u,i,[0])} = \mathsf{Adv}_3^{\mathbf{C2.2}}$, $\mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EM}}(u,i,[0])} = \mathsf{Adv}_3^{\mathbf{C2.3}}$, $\mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EE}}(u,i,[0],[0])} = \mathsf{Adv}_3^{\mathbf{C2.4}}$. Thus:

$$\mathsf{Adv}_3^{\mathbf{C4.1.1}} \leq 1/q + \mathsf{Adv}_3^{\mathbf{C2.1}} + \mathsf{Adv}_3^{\mathbf{C2.2}} + \mathsf{Adv}_3^{\mathbf{C2.3}} + \mathsf{Adv}_3^{\mathbf{C2.4}} + \epsilon_{\mathrm{GDH}}$$

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

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 $\mathsf{clean}_{\mathrm{EE}}$ implies the existence of a unique session at Bob with the same sid as that of Alice's session, refer to Fig. 18. As before, an adversary which could distinguish this game from its predecessor allows us to answer the Gap-Diffie–Hellman (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:

$$\mathsf{Adv}_3^{\mathbf{C4.1},\mathsf{clean}_{\mathrm{EE}}(u,i,x-1,x-1)} \leq 1/q + \epsilon_{\mathrm{GDH}}$$

*C4.2: Asymmetric Ratcheting, $s = [$**asym-ri**$:x]$, $x > 1$, $type[s] =$ `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:

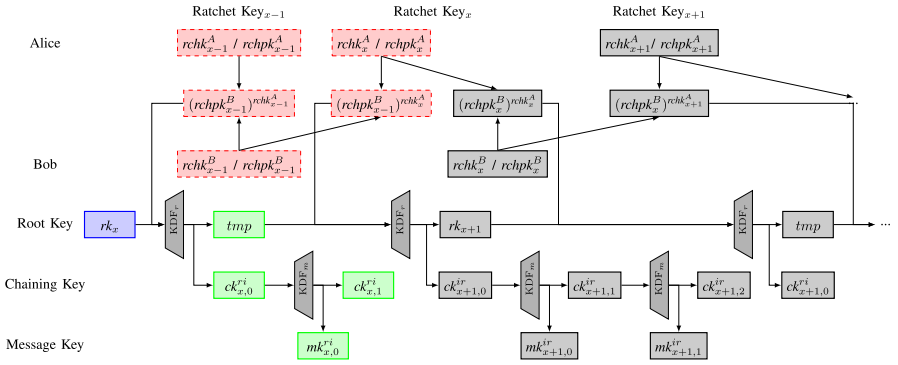



**Fig. 19.** A diagram showing the replacement of secrets in Game 7' of Case 4.2.1. In particular, [ ] denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); [ ] denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; [ ] denotes secrets that the challenger is able to replace with random, thus ensuring security; and [ ] denotes secrets that are not relevant to this case. Diagram legend can be found in Fig. 10 (Color figure online).

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 $E^{\texttt{asym-ri}}_{\text{clean-cur}}$: $\mathsf{clean}_{\text{EE}}(u, i, x, x-1)$

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

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, refer to Fig. 19. 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 $mk^{ri}_{x,0}$ as well as symmetric chaining keys $ck^{ri}_{x,0}$ and $ck^{ri}_{x,1}$ and intermediate value $tmp$, are all also then indistinguishable from random. For readability, we denote $\mathsf{Adv}_3^{\mathbf{C4.2},\mathsf{clean}_{\mathsf{prev}}} = \mathsf{Adv}_3^{\mathbf{C4.2.1}}$, $\mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\text{LM}}(u,i)} = \mathsf{Adv}_3^{\mathbf{C2.1}}$, $\mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\text{EL}}(u,i,[0])} = \mathsf{Adv}_3^{\mathbf{C2.2}}$, $\mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\text{EM}}(u,i,[0])} = \mathsf{Adv}_3^{\mathbf{C2.3}}$, $\mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\text{EE}}(u,i,[0],[0])} = \mathsf{Adv}_3^{\mathbf{C2.4}}$. Thus:

$$\mathsf{Adv}_3^{\mathbf{C4.2.1}} \leq 1/q + \mathsf{Adv}_3^{\mathbf{C2.1}} + \mathsf{Adv}_3^{\mathbf{C2.2}} + \mathsf{Adv}_3^{\mathbf{C2.3}} + \mathsf{Adv}_3^{\mathbf{C2.4}} + \epsilon_{\text{GDH}}$$

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

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

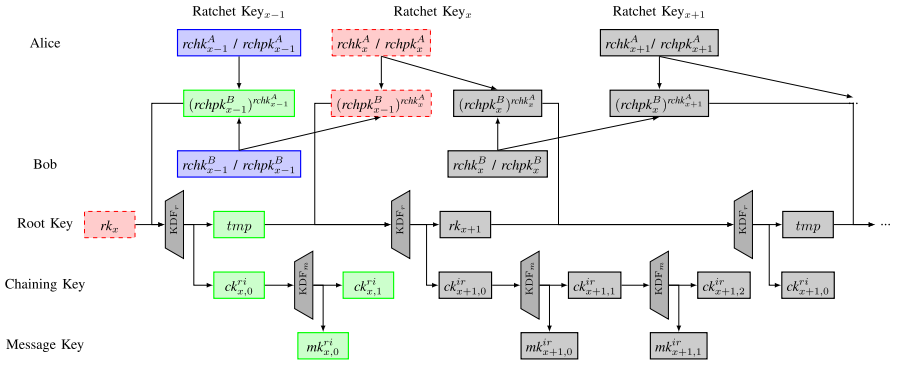



**Fig. 20.** A diagram showing the replacement of secrets in Game 3 of Case 4.2.2. In particular, denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); denotes secrets that are replaced with the output of a Test query from a Case 1 or Case 2 challenger; denotes secrets that the challenger is able to replace with random, thus ensuring security; and denotes secrets that are not relevant to this case. Diagram legend can be found in Fig. 10 (Color figure online).

$rchk^u_x$, $rchk^v_{x-1}$ of the Test session and its matching peer with values from a GDH challenger, refer to Fig. 20. 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:

$$\mathsf{Adv}_3^{\mathbf{C4.2},\mathsf{clean}_{\mathrm{EE}}(u,i,x-1,x-1)} \leq 1/q + \epsilon_{\mathrm{GDH}}$$

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

We finally arrive at the case of Signal in the multi-stage 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 symmetric stage. Thus,

$$\mathsf{Adv}_3^{\mathbf{C5}} \leq \mathsf{Adv}_3^{\mathbf{C5},\texttt{sym-ir}} + \mathsf{Adv}_3^{\mathbf{C5},\texttt{sym-ri}}$$

In all subcases, we will show that the probability of winning is 1/2.

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

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

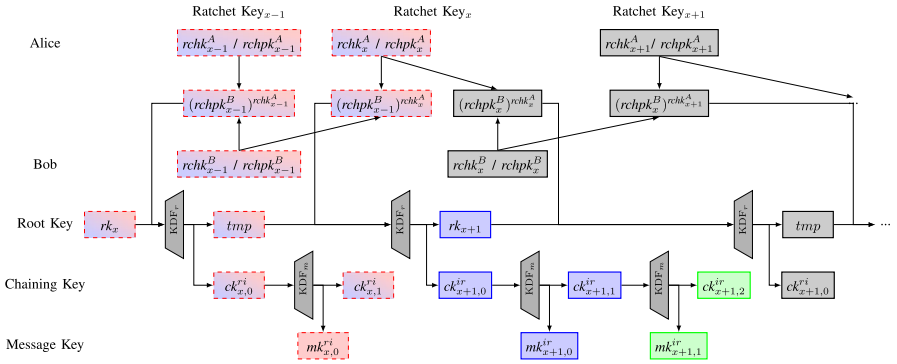



**Fig. 21.** A diagram showing the replacement of secrets in Game 7' of Case 5.1.2. All other subcases follow a similar replacement strategy. In particular, denotes secrets that the adversary may compromise via Reveal queries (or by computing the secrets as a result of the Reveal query); denotes secrets 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 $\mathcal{A}$. Diagram legend can be found in Fig. 10 (Color figure online).

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 $\mathsf{clean}_{\mathsf{state}}$). Thus:

$$\mathsf{Adv}_3^{\mathbf{C5},\mathtt{sym\text{-}ir}} \leq \mathsf{Adv}_3^{\mathbf{C5},\mathtt{sym\text{-}ir},x=0,y=1} + \mathsf{Adv}_3^{\mathbf{C5},\mathtt{sym\text{-}ir},x>0,y=1} + \mathsf{Adv}_3^{\mathbf{C5},\mathtt{sym\text{-}ir},x\geq 0,y>1}$$

(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* $s = [\textbf{\textit{sym-ir}}{:}x, y]$, $x = 0$, $y = 1$, $type[s] = \mathtt{sym\text{-}ir}$

This stage's messaging key is derived by applying a $\mathrm{KDF_m}$ to $ck^{ir}_{0,1}$, which was derived during the initial `triple` or `triple+DHE` handshake. By Case 3.2, $ck^{ir}_{0,1}$ is indistinguishable from random, refer to Fig. 21. Like the argument in Case 3.2, treating $\mathrm{KDF_m}$ as a random oracle, this stage's messaging key and the next chaining key $ck^{ir}_{0,2}$ are thus indistinguishable from random. Thus,

$$\mathsf{Adv}_3^{\mathbf{C5},\mathtt{sym\text{-}ir},x=0,y=1} \leq \mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{LM}}(u,i)} + \mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EL}}(u,i,[:0,])} + \mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EM}}(u,i,[:0,])} + \mathsf{Adv}_3^{\mathbf{C2},\mathsf{clean}_{\mathrm{EE}}(u,i,[:0,],[:0,])}$$

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

This stage's messaging key is derived by applying a $\text{KDF}_\text{m}$ to $ck^{ir}_{x,1}$, which was derived during asymmetric second-stage [**asym-ir**:$x$]. By Case 4.1, $ck^{ir}_{x,1}$ is indistinguishable from random. Like the argument in Case 2, treating $\text{KDF}_\text{m}$ as a random oracle, this stage's messaging key and the next chaining key $ck^{ir}_{x,2}$ are thus indistinguishable from random. Thus,

$$\mathsf{Adv}_3^{\mathbf{C5},\texttt{sym-ir},x>0,y=1} \leq \mathsf{Adv}_3^{\mathbf{C4.1},\mathsf{clean}_{\mathsf{prev}}} + \mathsf{Adv}_3^{\mathbf{C4.1},\mathsf{clean}_{\text{EE}}(u,i,x-1,x-1)}$$

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

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

$$\mathsf{Adv}_3^{\mathbf{C5},\texttt{sym-ir},x\geq 0,y>1} \leq \mathsf{Adv}_3^{\mathbf{C5},\texttt{sym-ir},x=0,y=1} + \mathsf{Adv}_3^{\mathbf{C5},\texttt{sym-ir},x>0,y=1}$$

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

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. Thus,

$$\mathsf{Adv}_3^{\mathbf{C5},\texttt{sym-ri}} = \mathsf{Adv}_3^{\mathbf{C5},\texttt{sym-ir}}$$

## 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 [9,39]. 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 [46], 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 [30,42], 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 Sect. 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 generalized security notion $lr$-PRF-ODH, which is parameterized 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 $\mathsf{ODH}_u$ and $\mathsf{ODH}_v$, respectively) where $\mathsf{n}$ indicates that no query is allowed, $\mathsf{s}$ indicates that a single query is allowed, and $\mathsf{m}$ indicates that arbitrarily many queries are allowed to the respective oracle. Consider the $lr$-PRF-ODH security experiment depicted in Fig. 22.

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

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

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.

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

1: $\mathsf{b} \xleftarrow{\$} \{0,1\}$, $c_u \leftarrow 0$, $c_v \leftarrow 0$
2: $u \xleftarrow{\$} \mathbb{Z}_q$, $\mathbf{X} \leftarrow \emptyset$
3: **if** $l = \mathsf{m}$ **then**
4: $\quad x^* \leftarrow \mathcal{A}(\mathbb{G}, g, q, g^u)^{\mathsf{ODH}_u}$
5: **else**
6: $\quad x^* \leftarrow \mathcal{A}(\mathbb{G}, g, q, g^u)$
7: $v \xleftarrow{\$} \mathbb{Z}_q$
8: **if** $\{g^{uv}, x^*\} \in \mathbf{X}$ **then**
9: $\quad$ **return** $\perp$
10: $y_0 \leftarrow \mathrm{PRF}(g^{uv}, x^*)$, $y_1 \xleftarrow{\$} \{0,1\}^\lambda$
11: $\mathbf{X} \leftarrow \mathbf{X} \cup \{g^{uv}, x^*\}$
12: $b' \leftarrow \mathcal{A}(y_b)^{\mathsf{ODH}_u, \mathsf{ODH}_v}$
13: **return** $(b' = b)$

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

1: $\mathsf{b} \xleftarrow{\$} \{0,1\}$, $c_u \leftarrow 0$, $c_v \leftarrow 0$
2: $u, v \xleftarrow{\$} \mathbb{Z}_q$, $\mathbf{X} \leftarrow \emptyset$
3: **if** $(l = \mathsf{m}) \wedge (r = \mathsf{m})$ **then**
4: $\quad \mathcal{A}(\mathbb{G}, g, q, g^u, g^v)^{\mathsf{ODH}_u, \mathsf{ODH}_v} \rightarrow x^*$
5: **if** $(l = \mathsf{m}) \wedge (r \neq \mathsf{m})$ **then**
6: $\quad \mathcal{A}(\mathbb{G}, g, q, g^u, g^v)^{\mathsf{ODH}_u} \rightarrow x^*$
7: **if** $(l \neq \mathsf{m}) \wedge (r = \mathsf{m})$ **then**
8: $\quad \mathcal{A}(\mathbb{G}, g, q, g^u, g^v)^{\mathsf{ODH}_v} \rightarrow x^*$
9: **else**
10: $\quad \mathcal{A}(\mathbb{G}, g, q, g^u, g^v) \rightarrow x^*$
11: **if** $(g^{uv}, x^*) \in \mathbf{X}$ **then**
12: $\quad$ **return** $\perp$
13: $y_0 \leftarrow \mathrm{PRF}(g^{uv}, x^*)$, $y_1 \xleftarrow{\$} \{0,1\}^\lambda$
14: $\mathbf{X} \leftarrow \mathbf{X} \cup \{g^{uv}, x^*\}$
15: $b' \leftarrow \mathcal{A}(y_b)^{\mathsf{ODH}_u, \mathsf{ODH}_v}$
16: **return** $(b' = b)$

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

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

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

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

**Fig. 22.** Security experiments for the generic PRF-ODH assumption, and the symmetric PRF-ODH assumption. Note that both experiments make use of identical $\mathsf{ODH}_u$ and $\mathsf{ODH}_v$ oracles.

**Definition 11.** (*Symmetric PRF-ODH assumption*) Let $\mathbb{G}$ be a cyclic group of order $q$ with generator $g$. Let $\mathrm{PRF} : \mathbb{G} \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$ 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 parameterized 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 $\mathsf{ODH}_u$ and $\mathsf{ODH}_v$, respectively) where $\mathsf{n}$ indicates that no query is allowed, $\mathsf{s}$ indicates that a single query is allowed, and $\mathsf{m}$ 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 Fig. 22. 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 initialized, we require many $\mathsf{ODH}_u$ and $\mathsf{ODH}_v$ 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 $\mathrm{KDF}_m$, 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 $\mathrm{KDF}_m$, or the mm-sPRF-ODH security of HKDF, $\mathbb{G}$.

**Case 1.2** In Game 6 of Case 1.2, we know that the predicate $\mathsf{clean}_{\mathrm{EL}}(u, i, [0])$ 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 initialized, we require many $\mathsf{ODH}_u$ 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 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 $\text{KDF}_m$, 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 $\text{KDF}_m$, or the mn-PRF-ODH security of HKDF, $\mathbb{G}$.

**Case 1.3** In Game 6 of Case 1.3, we know that the predicate $\mathsf{clean}_{\text{EM}}(u, i, [0])$ 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 initialized, we require many $\mathsf{ODH}_v$ 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 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 $\text{KDF}_m$ 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 $\text{KDF}_m$, or the mn-PRF-ODH security of 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 $\mathsf{clean}_{\text{EE}}(u, i, [0])$ 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 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 $\text{KDF}_m$ 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 $\text{KDF}_m$, or the sn-PRF-ODH security of HKDF, $\mathbb{G}$.

**Case 3.1** In Game 6 of Case 3.1, we know that the predicate $\mathsf{clean}_{\text{EE}}(u, i, [\textbf{asym-ri}{:}1])$ 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 $ck^{ri}_{1,0}$ 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 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 $x$, $ck^{ri}_{1,0}$ values in both the Tested session and its peer session that is used in computing the message key $mk^{ri}_{1,0}$ which was Tested by the adversary. In order to ensure that the message key $mk^{ri}_{1,0}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk^{ri}_{1,0}$ from $\text{KDF}_m$ and use the output from the PRF challenger to replace $mk^{ri}_{1,0}$. Thus, an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of $\text{KDF}_m$, or the sn-PRF-ODH security of HKDF, $\mathbb{G}$.

**Case 3.2** In Game 6 of Case 3.2, we know that the predicate $\mathsf{clean}_{\pi^i_u.type[:0,]}(u, i, [:0,])$ 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 $x$, $ck^{ri}_{1,0}$ values in both the Tested session and its peer session that is used in computing the message key $mk^{ri}_{1,0}$ which was Tested by the adversary. In order to ensure that the message key $mk^{ri}_{1,0}$ is indistinguishable from random, we need an additional PRF game to replace the computation of $mk^{ri}_{1,0}$ from $\text{KDF}_m$ and use the output from the PRF challenger to replace $mk^{ri}_{1,0}$. Thus, an adversary capable of distinguishing these changes would also be capable of breaking the PRF security of

$\text{KDF}_m$, or the mm-sPRF-ODH security of HKDF, $\mathbb{G}$ (as in Case 1.1), or the mn-PRF-ODH security of HKDF, $\mathbb{G}$ (as in Cases 1.2 and 1.3), or the sn-PRF-ODH security of 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 $\mathsf{clean}_{\texttt{asym-ri}}(u, i, [\textbf{asym-ri}{:}x - 1])$ 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 $\text{KDF}_m$, or the mm-sPRF-ODH security of HKDF, $\mathbb{G}$ (as in Case 1.1), or the mn-PRF-ODH security of HKDF, $\mathbb{G}$ (as in Cases 1.2 and 1.3), or finally the sn-PRF-ODH security of 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 $\mathsf{clean}_{\texttt{EE}}(u, i, x - 1, x - 1)$ 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 $\text{KDF}_r$ and $\text{KDF}_m$, or the sn-PRF-ODH security of HKDF, $\mathbb{G}$.

**Case 4.2.1** In Game 6 of Case 4.1.1, we know that the predicate $\mathsf{clean}_{\texttt{asym-ri}}(u, i, [\textbf{asym-ir}{:}x - 1])$ 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 $\mathsf{clean}_{\texttt{EE}}(u, i, x - 1, x - 1)$ 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 $\text{KDF}_r$ and $\text{KDF}_m$, or the sn-PRF-ODH security of 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 $\mathsf{clean}_{\texttt{sym}}$ 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) requires the strong symmetric variant of PRFODH (i.e. mm-PRF-ODH), whereas Case 2.4 (and similarly, Case 3.1) requires the weak non-symmetric variant 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 $\mathsf{ODH}_u$ oracle, but no queries to the $\mathsf{ODH}_v$ 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.

## References


[1] J. Alwen, S. Coretti, Y. Dodis, The Double Ratchet: security notions, proofs, and modularization for the Signal protocol, in *IACR Cryptology ePrint Archive 2018* (2018), p. 1037. https://eprint.iacr.org/2018/1037

[2] C. Bader, D. Hofheinz, T. Jager, E. Kiltz, Y. Li, Tightly-secure authenticated key exchange, in *TCC 2015, Part I*, LNCS, vol. 9014. (Springer, Heidelberg, 2015), pp. 629–658

[3] C. Bader, T. Jager, Y. Li, S. Schäge, *On the Impossibility of Tight Cryptographic Reductions*. Cryptology ePrint Archive, Report 2015/374 (2015). http://eprint.iacr.org/2015/374

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

[5] M. Bellare, A. Boldyreva, A. Palacio, An uninstantiable random-oracle-model scheme for a hybrid-encryption problem, in *Advances in Cryptology-EUROCRYPT 2004* (Springer, 2004), pp. 171–188

[6] M. Bellare, R. Canetti, H. Krawczyk, A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract), in *30th ACM STOC*. (ACM Press, 1998), pp. 419–428

[7] M. Bellare, D. Pointcheval, P. Rogaway, Authenticated key exchange secure against dictionary attacks, in *EUROCRYPT 2000*, LNCS, vol. 1807 (Springer, Heidelberg, 2000), pp. 139–155

[8] M. Bellare, P. Rogaway, Entity authentication and key distribution, in *CRYPTO'93.*, LNCS, vol. 773 (Springer, Heidelberg, 1994), pp. 232–249

[9] M. Bellare, P. 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] M. Bellare, A.C. Singh, J. Jaeger, M. Nyayapati, I. Stepanovs, *Ratcheted encryption and key exchange: the security of messaging*. Cryptology ePrint Archive, Report 2016/1028 (2016). http://eprint.iacr.org/2016/1028

[11] M. Bellare, B.S. Yee, Forward-security in private-key cryptography, in *CT-RSA 2003*, LNCS, vol. 2612 (Springer, Heidelberg, 2003), pp. 1–18

[12] D.J. Bernstein, Curve25519: new Diffie–Hellman speed records, in *PKC 2006*, LNCS, vol. 3958 (Springer, Heidelberg, 2006), pp. 207–228

[13] D.J. Bernstein, N. Duif, T. Lange, P. Schwabe, B.-Y. Yang, High-speed high-security signatures, in *CHES 2011*, LNCS, vol. 6917 (Springer, Heidelberg, 2011), pp. 124–142

[14] K. Bhargavan, C. Brzuska, C. Fournet, M. Green, M. Kohlweiss, S. Zanella-Béguelin, Downgrade resilience in key-exchange protocols, in *2016 IEEE Symposium on Security and Privacy* (IEEE Computer Society Press, 2016), pp. 506–525

[15] D. Bogado, D. O'Brien, *Punished for a Paradox*. Mar. 2, 2016. https://www.eff.org/deeplinks/2016/03/punished-forparadox-brazils-facebook (visited on 07/2016)

[16] N. Borisov, I. Goldberg, E. Brewer, Off-the-record communication, or, why not to use PGP, in *WPES* (ACM, Washington DC, 2004), pp. 77–84

[17] C. Boyd, C. Cremers, M. Feltz, K.G. Paterson, B. Poettering, D. Stebila, ASICS: authenticated key exchange security incorporating certification systems, in *ESORICS 2013*, LNCS, vol. 8134 (Springer, Heidelberg, 2013), pp. 381–399

[18] J. Brendel, M. Fischlin, F. Günther, C. Janson, PRF-ODH: relations, instantiations, and impossibility results, in *CRYPTO 2017, Part III* LNCS, vol. 10403 (Springer, Heidelberg, 2017), pp. 651–681

[19] R. Canetti, O. Goldreich, S. Halevi, The random oracle methodology, revisited, in *Journal of the ACM (JACM)* 51.4 (2004), pp. 557–594

[20] R. Canetti, S. Halevi, J. Katz, A forward-secure public-key encryption scheme, in *EUROCRYPT 2003*, LNCS, vol. 2656 (Springer, Heidelberg, 2003), pp. 255–271
[21] R. Canetti, H. Krawczyk, Analysis of key-exchange protocols and their use for building secure channels, in *EUROCRYPT 2001*, LNCS, vol. 2045 (Springer, Heidelberg, 2001), pp. 453–474
[22] K. Cohn-Gordon, C. Cremers, *Mind the Gap: Where Provable Security and Real-World Messaging Don't Quite Meet*. Cryptology ePrint Archive, Report 2017/982 (2017). http://eprint.iacr.org/2017/982
[23] K. Cohn-Gordon, C. Cremers, L. Garratt, *On Post-Compromise Security*. (A shorter version of this paper appears at CSF 2016) (2016). http://eprint.iacr.org/2016/221
[24] Conversations. https://conversations.im/ (visited on 07/2016)
[25] C. Cremers, M. Feltz, *One-round Strongly Secure Key Exchange with Perfect Forward Secrecy and Deniability*. Cryptology ePrint Archive, Report 2011/300 (2011). http://eprint.iacr.org/2011/300
[26] C. Cremers, M. Horvat, S. Scott, T. 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, 2016)
[27] J.P. Degabriele, A. Lehmann, K.G. Paterson, N.P. Smart, M. Strefler, On the joint security of encryption and signature in EMV, in *CT-RSA 2012*, LNCS, vol. 7178 (Springer, Heidelberg, 2012), pp. 116–135
[28] M. Di Raimondo, R. Gennaro, H. Krawczyk, Deniable authentication and key exchange, in *ACM CCS 2006* (ACM Press, 2006), pp. 400–409
[29] M. Di Raimondo, R. Gennaro, H. Krawczyk, Secure off-the-record messaging, in *WPES*. (ACM, Alexandria, VA, 2005), pp. 81–89
[30] B. Dowling, M. Fischlin, F. Günther, D. Stebila, A cryptographic analysis of the TLS 1.3 handshake protocol candidates, in *ACM CCS 2015* (ACM Press, 2015), pp. 1197–1210
[31] F. Betül Durak, S. Vaudenay, Bidirectional asynchronous ratcheted key agreement without key-update primitives, in *IACR Cryptology ePrint Archive 2018* (2018), p. 889. https://eprint.iacr.org/2018/889
[32] Electronic Frontier Foundation, Secure messaging scorecard (2016). https://www.eff.org/node/82654
[33] Facebook. Messenger Secret Conversations, Technical report (2016). https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf (visited on 07/2016)
[34] M. Fischlin, F. Gúnther, Multi-stage key exchange and the case of Google's QUIC protocol, in *ACM CCS 2014* (ACM Press, 2014), pp. 1193–1204
[35] T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, T. Holz, *How Secure is TextSecure?* Cryptology ePrint Archive, Report 2014/904 (2014). http://eprint.iacr.org/2014/904 (Version from April 5, 2016)
[36] T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, T. Holz, How secure is TextSecure?, in *1st IEEE European Symposium on Security and Privacy* (IEEE Computer Society Press, 2016)
[37] C. Garman, M. Green, G. Kaptchuk, I. Miers, M. Rushanan, Dancing on the lip of the volcano: chosen ciphertext attacks on Apple iMessage, in *Usenix Security 2016* (2016)
[38] Wire Swiss GmbH. Wire Security Whitepaper, in (Aug. 17, 2018). https://wire-docs.wire.com/download/Wire+Security+Whitepaper.pdf (visited on 01/2019)
[39] S. Goldwasser, Y.T. Kalai, Cryptographic assumptions: a position paper, in *IACR Cryptology ePrint Archive 2015* (2015), p. 907
[40] M.D. Green, I. Miers, Forward secure asynchronous messaging from puncturable encryption, in *2015 IEEE Symposium on Security and Privacy* (IEEE Computer Society Press, 2015), pp. 305–320
[41] M. Hamburg, *Ed448-Goldilocks, a New Elliptic Curve*. Cryptology ePrint Archive, Report 2015/625 (2015). http://eprint.iacr.org/2015/625
[42] T. Jager, F. Kohlar, S. Schäge, J. Schwenk, On the security of TLS-DHE in the standard model, in *CRYPTO 2012*, LNCS, vol. 7417 (Springer, Heidelberg, 2012), pp. 273–293
[43] T. Jager, J. Schwenk, J. Somorovsky, On the security of TLS 1.3 and QUIC against weaknesses in PKCS#1 v1.5 Encryption, in *ACM CCS 2015* (ACM Press, 2015), pp. 1185–1196
[44] D. Jost, U. Maurer, M. Mularczyk, Efficient ratcheting: almost-optimal guarantees for secure messaging, in *IACR Cryptology ePrint Archive 2018* (2018), p. 954. https://eprint.iacr.org/2018/954
[45] N. Kobeissi, K. Bhargavan, B. 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, 2017)
[46] N. Koblitz, A.J. Menezes, The random oracle model: a twenty-year retrospective, in *Designs, Codes and Cryptography 77.2-3* (2015), pp. 587–610

[47] M. Kohlweiss, U. Maurer, C. Onete, B. Tackmann, D. Venturi, (De-)constructing TLS 1.3, in *INDOCRYPT 2015*, LNCS, vol. 9462 (Springer, Heidelberg, 2015), pp. 85–102
[48] H. Krawczyk, Cryptographic extraction and key derivation: the HKDF scheme, in *CRYPTO 2010*, LNCS, vol. 6223 (Springer, Heidelberg, 2010), pp. 631–648
[49] H. Krawczyk, HMQV: a high-performance secure Diffie–Hellman protocol, in *CRYPTO 2005*, LNCS, vol. 3621 (Springer, Heidelberg, 2005), pp. 546–566
[50] C. Kudla, K.G. Paterson, Modular security proofs for key agreement protocols, in *ASIACRYPT 2005*, LNCS, vol. 3788 (Springer, Heidelberg, 2005), pp. 549–565
[51] B.A. LaMacchia, K. Lauter, A. Mityagin, Stronger security of authenticated key exchange, in *ProvSec 2007*, LNCS, vol. 4784 (Springer, Heidelberg, 2007), pp. 1–16
[52] A. Langley. Pond. (2014). https://pond.imperialviolet.org/ (visited on 06/22/2015)
[53] X. Li, J. Xu, Z. Zhang, D. Feng, H. Hu, Multiple handshakes security of TLS 1.3 candidates, in *2016 IEEE Symposium on Security and Privacy*. (IEEE Computer Society Press, 2016), pp. 486–505
[54] `libsignal-protocol-java`. GitHub repository, commit hash `4a7bc1667a68c1d8e6af0151be30b84b94fd1e38` (2016). https://github.com/WhisperSystems/libsignal-protocol-java (visited on 07/2016)
[55] M. Marlinspike, *Advanced Cryptographic Ratcheting. Blog. 2013*. https://whispersystems.org/blog/advanced-ratcheting/ (visited on 07/2016)
[56] A. Menezes, B. Ustaoglu, On reusing ephemeral keys in Diffie–Hellman key agreement protocols, in *Int. J. Appl. Cryptol.* 2.2 (2010), pp. 154–158
[57] V. Moscaritolo, G. Belvin, P. Zimmermann, *Silent Circle Instant Messaging Protocol Specification*. Technical report Archived from the original. Dec. 5, 2012. https://web.archive.org/web/20150402122917/, https://silentcircle.com/sites/default/themes/silentcircle/assets/downloads/SCIMP_paper.pdf (visited on 07/2016)
[58] T. Okamoto, D. Pointcheval, The Gap-problems: a new class of problems for the security of cryptographic schemes, in *PKC 2001*, LNCS, vol. 1992 (Springer, Heidelberg, 2001), pp. 104–118
[59] K.G. Paterson, J.C.N. Schuldt, M. Stam, S. Thomson, On the joint security of encryption and signature, Revisited, in *ASIACRYPT 2011*, LNCS, vol. 7073 (Springer, Heidelberg, 2011), pp. 161–178
[60] T. Perrin, *Double Ratchet Algorithm*. GitHub wiki (2016). https://github.com/trevp/double_ratchet/wiki (visited on 07/22/2016)
[61] T. Perrin, *The XEdDSA and VXEdDSA Signature Schemes*. Specification (2016). https://whispersystems.org/docs/specifications/xeddsa/ (visited on 07/2016)
[62] T. Perrin, M. Marlinspike, *The Double Ratchet Algorithm*. Specification (2016). https://whispersystems.org/docs/specifications/doubleratchet/ (visited on 01/2017)
[63] T. Perrin, M. Marlinspike, *The X3DH Key Agreement Protocol*. Specification (2016). https://whispersystems.org/docs/specifications/x3dh/ (visited on 01/2017)
[64] B. Poettering, P. Rösler, Towards bidirectional ratcheted key exchange, in *Advances in Cryptology—CRYPTO 2018—38th Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 19–23, 2018, *Proceedings, Part I*, Lecture Notes in Computer Science, vol. 10991 (Springer, 2018), pp. 3–32. ISBN:978-3-319-96883-4. https://doi.org/10.1007/978-3-319-96884-15C_1
[65] J. Reardon, D. Basin, S. Capkun, SoK: secure data deletion, in *2013 IEEE Symposium on Security and Privacy (SP)*, (2013), pp. 301–315
[66] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*. Internet-Draft draft-ietf-tls-tls13–14 (2016). http://www.ietf.org/internet-drafts/draft-ietf-tls-tls13-14.txt
[67] P. Rogaway, Authenticated-encryption with associated-data, in *ACM CCS 2002* (ACM Press, 2002), pp. 98–107
[68] P. Rösler, C. Mainka, J. Schwenk, More is less: on the end-to-end security of group chats in Signal, WhatsApp, and Threema, in *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018*, (London, UK, April 24–26), 2018 (IEEE, 2018), pp. 415–429. ISBN:978-1-5386-4228-3. https://doi.org/10.1109/EuroSP.2018.00036
[69] A. Straub, *OMEMO Encryption*. Oct. 25, 2015. https://conversations.im/xeps/multi-end.html (visited on 07/2016)
[70] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, M. Smith, SoK: secure messaging, in *2015 IEEE Symposium on Security and Privacy* (IEEE Computer Society Press, 2015), pp. 232–249

[71] N. Unger, I. Goldberg, Deniable key exchanges for secure messaging, in *ACM CCS 2015* (ACM Press, 2015), pp. 1211–1223
[72] WhatsApp, *WhatsApp Encryption Overview*. Technical report (2016). https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf (visited on 07/2016)