Secure Messaging Authentication against Active Man-in-the-Middle Attacks

Benjamin Dowling
Department of Computer Science
ETH Zürich
Zürich, Switzerland
benjamin.dowling@inf.ethz.ch

Britta Hale
Computer Science Department
Naval Postgraduate School (NPS)
Monterey, USA
britta.hale@nps.edu

2021 IEEE European Symposium on Security and Privacy (EuroS&P) | 978-1-6654-1491-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/EUROSP51992.2021.00015

***Abstract*—Modern messaging applications often rely on out-of-band communication to achieve entity authentication, with human users verifying and attesting to long-term public keys. This is done primarily to reduce reliance on trusted third parties by replacing that role with the user. Despite a great deal of research focusing on analyzing the *confidentiality* aspect of secure messaging, the *entity authenticity* aspect of it, which relies on the user mediation, has been largely assumed away. Consequently, while many existing protocols provide some confidentiality guarantees after a compromise, such as post-compromise security (PCS), authenticity guarantees are generally lost, especially against an active attacker. This leads to potential man-in-the-middle (MitM) attacks. In this work, we address this gap by proposing a model to formally capture user-mediated entity authentication, as used by real-world protocols, that can be composed with any ratcheted key exchange. Our threat model captures active post-compromise entity authentication security. We demonstrate that the Signal application's user-mediated authentication protocol cannot be proven secure in this model and suggest a straightforward fix for Signal that allows the detection of an active adversary. Our results have direct implications for other existing and future ratcheted secure messaging applications.**

***Index Terms*—Secure Messaging, Ratcheted Authentication, Signal, User-Mediated Authentication, Ceremonies**

1. Introduction

Ratcheted key exchange protocols have gained extreme popularity in recent years due to the strong security guarantees achievable in such designs, including forward secrecy (FS) and post-compromise security (PCS) for *confidentiality*. The Signal protocol is one example, being one of the most ubiquitous cryptographic protocols used in practice, it enables end-to-end encryption for widespread secure messaging applications including WhatsApp [35], Wire [14], Skype [31], and Facebook Messenger Secret Conversations [11]. On a high-level, these are two-party multi-stage key exchange protocols, where parties repeatedly exchange "update" information to derive independent and secret symmetric keys. Despite the popularity of ratcheted key exchange protocols, entity authentication – a critical pillar of secure communication – has been fundamentally abstracted away in their analysis. This has left a critical gap in the understanding of ratcheted key exchange protocol security.

Consider the case of mobile device access, such as is common at border crossings [1], [26], [33]. By temporarily accessing device state, an adversary can learn both current encryption and signature keys, and read data. After the device has been returned, the owner sends a legitimate message to ratchet the state forward, being assured by the property of PCS that the attacker is now locked out – *if the attacker does nothing*. If, however, the attacker is *active*, it may send an update message before returning the device to its owner, or in any case before the owner sends its next message. That simple action prevents traditional PCS and allows the adversary to act as a man-in-the-middle (MitM) *indefinitely and without detection*. Moreover, since state leakage has also allowed access to signature keys that are used to bootstrap security for each session (chat thread), the owner cannot simply restart sessions to prevent this. An attacker that can impersonate a party or modify updates undetected can also break confidentiality and data authenticity. To put it succinctly: an active attacker can exploit the lack of meaningful entity authentication in the "after compromise" security threat model to subvert other channel security guarantees.

With such implications, it may seem surprising that entity authentication against active attackers has been largely overlooked in protocol analyses [4], [8], [27]. In practice, this omission is due to the out-of-band (OOB) nature of the entity authentication that is usually employed and the challenge of analyzing it systematically. Signal and similar protocols rely heavily on *user mediation*, where users actively engage in the verification of long-term keys, effectively replacing a trusted third party or certificate authority. Capturing this user-mediated authentication in a ratcheted setting necessitates a new model – one that can handle unpredictable user behavior and the device interfaces.

In this work, we investigate user mediation in the ratcheted key environment, formalising a computational model for user-mediated protocol analysis. Our model builds on previous work to capture active post-compromise (APCS) security with respect to entity authentication (namely PCS against an active attacker). We apply this model to Signal, and note that the Signal authentication mechanism does not satisfy our security requirements. Finally, we present a straight-forward solution for the Signal ratcheted key exchange protocol that can detect active attackers.

Now we briefly introduce the surrounding context, including the Signal authentication protocol, the problem

DOI 10.1109/EuroSP51992.2021.00015

of APCS security, and the practical use and analysis of user mediation in security protocols.

Signal Protocol

The Signal protocol (or simply "Signal") [19] is a ratcheted key exchange and encryption protocol which has become synonymous with secure messaging. Mutual entity authentication in Signal is performed through user interaction via the Signal authentication protocol: a user manually compares "safety numbers" on both devices, either through visual comparison or a QR code reader, and finishes by selecting "verified" on each device. At any time the user may also deselect the "verified" status of the conversation. In addition to the Signal app, this mechanism for entity authentication is used in Wire [13], WhatsApp [35], and Facebook Messenger Secret Conversations [11].

Compared to research on the Signal ratcheted key exchange, the Signal authentication protocol has been left almost entirely unconsidered. Aside from a comparative usability study on Signal's safety numbers [5], Signal authentication protocol has not been formally treated in the literature focused on the Signal protocol [2], [6]–[8], [28], with focus instead being on FS and PCS *confidentiality* guarantees following device compromise. The Signal specification itself notes the problematic consequence of a compromise scenario on security, calling it "disastrous", and specifically stating for authentication that loss of identity keys "allows impersonation of that party" [20].

Post-Compromise Security

PCS (also known as *future secrecy* or *self-healing*) [9] is a recent security notion that formally captures a protocol's ability to "lock out" an attacker following full state compromise, contingent on the attacker being passive for some period. To achieve PCS, protocols such as Signal use a technique known as *key ratcheting*. Ratcheting protocols first establish a shared "root" secret (which we denote rk) that is then continually ratcheted forward at each *epoch*. This is done using a keyed function f that takes as input the current root key rk^i and additional entropy ek^i (i.e. $rk^{i+1} \leftarrow f(rk^i, ek^i)$). Root secrets can be used to derive further keys that will be used in an arbitrary symmetric-key protocol. Note that an epoch is defined as the period between the derivation of two root secrets. It follows that even if an attacker has exposed the root secret rk^i, if the communicating parties are able to secretly establish new entropy ek^i in that epoch, then the attacker cannot compute the following root secret rk^{i+1}.

Typically, PCS is considered only in terms of confidentiality, where an attacker will not be able to read future messages following a compromise. However confidentiality only represents a partial view of security in the *post-compromise threat model* (i.e. following a compromise). We extend the view of the post-compromise threat model to include entity authentication active attacks – not only by healing from a passive attacker but also detect active impersonation.

Consider the following scenario to demonstrate why this matters: a user wishes to verify that their session is not compromised by an active attacker injecting messages between the two devices, and decides to use the Signal entity authentication protocol. A successful verification indicates that an attacker has not modified the long-term keys of either party in the session. However, as described in Section 2.3, the verification only authenticates *long-term keys* and not *current session-specific* state. This design choice was for the sake of usability [30], but it still weakens the security guarantees of the protocol. Following a compromise, an attacker can send/receive messages and impersonate a partner device, and this will not be detected after an execution of the Signal entity authentication protocol. Thus, under a standard model of communication (i.e. two devices and a single channel) and following a full state compromise, users *must* rely on the attacker to be passive in order to recover security – something that is not assured.

User Mediated Authentication

Traditionally, cryptographic analysis considers only device-to-device communication, with all user interactions OOB. However, such a perspective leads to erroneous security assumptions on user-input data and honest user behaviour. In practice, a user reads information from a device display and acts accordingly – an attacker may eavesdrop on the user input data, the display data, or may even obtain access to the device. Using malware, an attacker may also display data via a display overlay, without having access to private keys. The Tap n' Ghost attack [23] is one such example, where the attacker affects both the user input to the device and display but has no access to internal keys or state, by externally injecting electrical noise that affects capacitive touch screens.

User mediated authentication was considered in [15] with a model variant being applied in the analysis of an ISO authentication protocol. In that model, an adversary is considered under three different capability levels, ranging from eavesdropping to device control. We extend the model of [15] for more fine-grained control, allowing unlimited adversarial modification of messages to a user via queries, but with freshness capturing such action. We separate out the user-to-device channel as two uni-directional channel controls to separately model an attacker's control of a device display and control of the user input (e.g. through social engineering). Moreover, we consider two security goals: authentication security under display and user compromise, and authentication security under device compromise (including all keys). Finally, this our model is specifically extended to handle ratcheted protocols, such that we can capture the PCS properties relevant to Signal. We denote this security framework the *Mediated Epoch Three-party Authentication* ($META$) model. $META$ has implications not only for the analysis of Signal, but also other user-mediated protocols such as Bluetooth.

$META$ is not an authenticated key exchange (AKE) model per se. In mainstream messaging applications [14], [19], [35] the ratcheted key exchange and messaging protocols are neatly composed, where per-epoch root secrets from the key exchange are input into the symmetric-key protocol for protection of actual messages (data). $META$ aims to capture authentication per ratchet for epoch secrets. We define two variants of ratcheting freshness and corresponding security experiments for ratcheting authentication, dependent on adversarial capabilities – under user compromise (CompUser) and under device compromise (CompDev) threat models.

CompUser formalises freshness against an attacker capable of controlling a user's actions and controlling information displayed to the user on a single device. This formalises security guarantees when one device may be

infected with malware, leading to control of a device display but no access to keys. It also captures shoulder-surfing attacks and the Tap n' Ghost mentioned above. When the user is an active protocol participant, any malicious application that can display content to the user can conceivably mirror a viable output and trick the user. One can think of this as a pop-up ad that is designed to look like the legitimate application. Such attacks are on the user interface and may not require access to internal application secrets (see e.g. the Strandhogg vulnerability [16]). CompDev security, in comparison, allows device state to be compromised but limits adversarial capabilities on the user-to-device channels. CompDev security is closely linked to tradition channel security experiments. Note that in both CompUser and CompDev security we allow an eavesdropping adversary that can read all information between a user and device (e.g. shoulder-surfing). It is under CompUser security that we capture APCS for entity authentication. Interestingly, the choice of session identifier defines whether satisfaction of CompUser security may imply authentication of the entire session up to that epoch (see Section 3).

Related Work

The concept of reaching beyond normal protocol interaction to consider use-context can also be seen in *ceremonies* [10], which can be seen as the OOB roles and actions of users in the context of security protocols, and aims to model and analyze traditionally out-of-scope aspects of these protocols, including user interactions and societal contexts [3]. The concept has been applied to public key infrastructure [22] and to verifiable elections [17]. Intrinsically, the modelling approach that ceremonies take differ from our approach in fundamental goals and assumptions – we build on computational models while ceremonies are based on the Dolev-Yao model. Ceremonies provide a simpler, but wider view of the environment, while we expand upon lines of research capturing complex secure messaging and user-mediated security frameworks.

Other works have considered usability aspects of such protocols [29], [34]. Usability is an important yet orthogonal issue to what we address here; namely, once designers have identified what they believe to be the most effective means of employing user interaction, our model can provide insight on the conditions under which the design is secure. For example, as will be shown, Signal employs a "trust on first use" paradigm in compromise-centric model where FS and PCS are prioritized over efficiency. Under this paradigm, participants have no initial authentication of the identities involved but have the option of authenticating via QR codes or safety numbers later. We show that Signal does not ever provide entity authentication under this paradigm since the authentication secrets are not linked to such eventual stages of the protocol where authentication would take place, despite compromise being an intrinsic assumption to the security model.

In complementary work, the behaviour of "lazy users" who do not compare full e.g. safety number strings within user mediated protocols such as Signal has also been examined [24]. In that work, the authors focus on a static (non-ratcheting) authentication protocol solution. While both of these research lines point to ways of capturing user behaviour in such protocols, our model is able to capture authentication within the ratcheted key exchange context and thereby formalize APCS.

Contributions

In this work, we address the incongruity of PCS without authentication against active attackers.

- We formalize security for ratcheted user-mediated authentication in the post-compromise threat model (under both CompUser and CompDev attacks), and introduce the $META$ model. In $META$ security, a user is guaranteed that, should a compromise occur, the protocol will either detect the attack or self-heal through normal PCS, thereby achieving APCS.
- We demonstrate that the Signal authentication protocol is unable to satisfy $META$ security.
- We propose a modification of the Signal key exchange and authentication protocols, called the *Modified Device-to-User Signal Authentication* (MoDUSA) protocol. As in the traditional Signal authentication protocol, a user may initiate the MoDUSA authentication check whenever and however often they desire.
- We prove that the MoDUSA protocol achieves our strong notion of $META$ security. By linking authentication to ratcheting epochs, MoDUSA provides authentication for all epochs in the session up through when MoDUSA is run.

2. The Signal Protocol

In this section we give an overview of the Signal Protocol, almost certainly the most widespread example of a two party asynchronous messaging protocol. Signal's structure allows users to send encrypted messages to each other with forward secrecy and post-compromise security. On a high level, the protocol consists of three distinct stages, where *Alice* and *Bob* refer to respective devices in the protocol (we refer to a single human *User* throughout (see Fig. 4 for example).

- A *session establishment* stage, where one user Alice fetches a prekey bundle belonging to another user Bob, and uses Bob's public keyshare information (as well as generating Alice's own secret keyshare values) to derive a *root key*, a *chain key*, and a *message key*. This is called the extended Triple Diffie-Hellman (X3DH) protocol [21], which we give an overview for here, and describe in detail in Section A.
- An *asymmetric ratcheting* stage, where a user Alice, after receiving a message from Bob containing a new *ratchet key*, generates a new *ratchet* key, performs a DH computation with Bob's ratchet, and uses the output and the current root key in a KDF to generate a new root key, a new chain key, and a new message key. This is an *asymmetric ratchet* of the Double Ratchet protocol [18],
- A *symmetric ratcheting* stage, where a user Alice, after sending a message to Bob decides to send another message to Bob, using the chain key to roll forward (using a key derivation function) and derive a new chain key and a new message key, to be used when the user sends a new message. This is the *symmetric ratchet* of the Double Ratchet protocol [18]. The symmetric ratchet is outside the scope for our analysis, and so we omit a description here.

2.1. Terminology

Here we introduce the terminology of Signal, as well as the notation that we use to describe its components. We describe all types of keys and shared secret values that are computed during a Signal protocol execution, as well as user authentication protocols.

- *identity keys* $(idk, idpk)$: Long-term DH keypairs used to sign other keys used in Signal, as well as derive keys in multiple X3DH key exchanges.
- *signed prekeys* $(spk, sppk)$: Medium-term DH keypairs, signed by the user, and used to derive keys in multiple X3DH key exchanges.
- *one-time-keys* $(otk, otpk)$: Ephemeral DH keypairs, used to derive keys in a single X3DH key exchange.
- *ratchet keys* $(rck, rcpk)$: Ephemeral DH keypairs, used to derive keys in both the X3DH key exchange, as well as asymmetric ratcheting stages.
- *root keys* (rk^i): A symmetric secret value, used to generate the i-th root and chain keys during an asymmetric ratchet stage, using the $(i\text{-}1)$-th and ith ratchet keys.
- *chain keys* (ck^i_j): A symmetric secret value derived from the $(i\text{-}1)$-th root key, used to generate the i-th chain and message keys during the j-th symmetric ratchet stage, with no added entropy.
- *message keys* (mk^i_j): A symmetric secret key, derived from the $(j\text{-}1)$-th chain key and used to AEAD-encrypt a plaintext message.
- *fingerprint* (`fprint`): a representation of some session identifying information, used by the human user to authenticate both parties.

2.2. Signal Key Exchange

In Signal, sessions are established in either an offline or online fashion. Either process uses *prekey bundles*, a set of keys that are generated locally by each user device and then either sent to a centralized Signal server (offline mode) or sent upon request to another user via the server (online mode). When a user Alice wishes to establish a session with an offline user Bob, Alice retrieves Bob's prekey bundle from the Signal server, and uses the values within to create shared secret values. This process of deriving keys from prekey bundles is referred to as the Extended Triple Diffie-Hellman (X3DH) key agreement protocol. The full details of the prekey bundle can be viewed in Section A.

After Alice fetches the prekey bundle, the signature σ_B over the signed prekey $sppk_B$ is verified using Bob's long-term identity key $idpk_B$. An important observation here is that Signal, unlike protocols such as Transport Layer Security, has no public key infrastructure for authenticating identity keys. Thus, any attacker that controls the communication channel (such as the Signal server) is capable of injecting its own identity key $idpk_{B'}$, use it to sign prekeys, and impersonate Bob. To prevent this attack, users can authenticate to each other using "fingerprints." This mechanism is supported within the Signal authentication protocol (discussed in Section 2.3) and is not a part of the Signal protocol itself.

Once a session has been established there are two different mechanisms to derive new message keys. The first is called *asymmetric ratcheting*, and is triggered the first time a user sends a message to their conversation partner after having received a message. The second is called *symmetric ratcheting*, and is triggered when the user sends another message in a chain, without having received a new message from their conversation partner.

Asymmetric ratcheting requires the user generate a new DH key pair $(rck^i, rcpk^i)$ called *ratchet keys*. This new ratchet key is used with the previous ratchet key $rcpk^{i-1}$ (from the conversation partner), as well as the currently maintained root key rk^{i-1} to derive a new root and chain key $rk^i, ck^i_0 \leftarrow \text{KDF}(rk^{i-1}, (rcpk^{i-1})^{rck^i})$. Since this new chain key will be used to generate message keys for the device until it receives a new message from its conversation partner, it also updates two counters $pctr, ctr$[1] such that $pctr \leftarrow ctr$, $ctr \leftarrow 0$. The device then computes the next chain key and the first message key in this chain $ck^i_1, mk^i_1 \leftarrow \text{KDF}(ck^i_0)$, and uses an AEAD symmetric cipher to encrypt a plaintext message $ptxt'$. The additional data field AD is set as the AD field from the first message in the conversation, as well as the sender's most recently generated ratchet key and its currently maintained counters, i.e. $AD = \{idpk_A \| idpk_B \| rcpk^0_A \| rcpk^i_I \| ctr \| pctr\}$. A protocol flow diagram is given in Fig. 8, Appendix A.

2.3. Entity Authentication in Signal

Here we introduce the user-mediated entity authentication mechanism used in the Signal application. The Signal app supports post-session establishment entity authentication via a human user interface. On a high level, the Signal app produces fingerprints of the long-term keys and identities of both parties in the communication channel. These fingerprints can take two distinct forms: QR code representation or numeric code representation, and below we describe how each is computed. Due to the lack of formal specification for the Signal authentication protocol, this description is guided by the implementation [25].

After a user validates their communication partner, the user can mark their communication partner as "verified." When the user proceeds, they are alerted when these fingerprints have changed - this may occur due to their communicating partner changing their device or identity key. Any further attempt at communication on the part of the user will require them to manually acknowledge that the message may be sent to an unverified partner.

Authentication with QR codes

The QR code verification method (which the repository [25] refers to as "scannable fingerprints") allows users to authenticate each other without the risk of human error while reading and comparing fingerprints. There are two different versions of generating these scannable fingerprints, `version 0` and `version 1`. For the rest of the paper we focus on the `version 1` method of generating scannable fingerprints, as it is both a most recent version, and is more closely related to the numeric code verification method (which the repository refers to as "displayable fingerprints"). While a QR code is verified out-of-band (OOB) by a user, the verification itself is error-free, assuming an honest QR code reader.

1. $pctr$ is a counter of messages sent using message keys generated from root key rk^{i-2}, and ctr is the number of messages that have been sent using message keys generated from root key rk^i

Scannable Fingerprint Version 1. In `version 1`, generating scannable fingerprints requires computing a digest of the *local* and *remote* identifiers and identity keys. Each device computes fingerprints as follows (here A acts as the local partner, and B the remote partner):

$$\texttt{local\_fprint} = H_i(0 \| \texttt{fvers}^2 \| idpk_A \| \mathrm{ID}_A, idpk_A)$$

$$\texttt{remote\_fprint} = H_i(0 \| \texttt{fvers} \| idpk_B \| \mathrm{ID}_B, idpk_B)$$

$H_i(x, y)$ is an iterative hash, where $H_0 = H(x)$, and $H_i = H(H_{i-1} \| y)$. In the repository we examined, the Signal authentication protocol uses SHA2 with 512-bits of output as the underlying hash function, with 5200 iterations. The fingerprint is a QR code representation of the following fields: {`svers`, 0, `local_fprint`, `remote_fprint`}. Afterwards, the device serializes the fingerprint and generates a QR code from the serialized data. The devices can then verify the communicating partner's scannable fingerprint as described above.

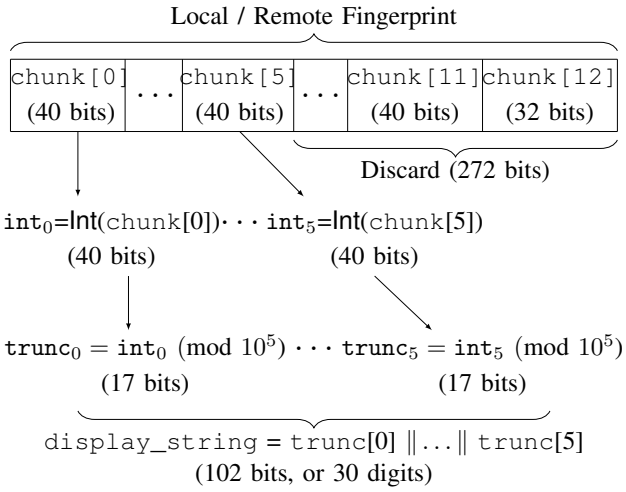

Figure 1. Truncation method used by Signal to reduce the length of the (local or remote) fingerprint output by the hash function to a human-readable state. Int() denotes a function that converts a bit-string to an integer value.

For numeric codes, the Signal authentication protocol first generates local fingerprint `local_fprint` and remote fingerprints `local_fprint` as described above.

Recall that fingerprint generation uses SHA2 with 512-bit output length as its underlying hash function. However, the Signal authentication protocol's numeric code, read by the users, is not the direct output of the hash function, but instead the concatenation of two 30-digit representations of truncated hash outputs, described in Fig. 1.

The displayed fingerprint read by the human users is then the sorted concatenation of the local and remote "`display_strings`", e.g.

$$\{\texttt{local\_display\_string}, \texttt{remote\_display\_string}\}$$

where sorting is according to the relative size of the users' public keys ($idpk$).

2. `fvers` here refers to the "fingerprint generation version". Note that in both QR and numeric codes, `fvers` = 0.

2.4. Vulnerabilities Against Authentication

We now discuss weaknesses and how active attackers are not detected by the Signal entity authentication protocol.

No session authentication. From Section 2.3 it follows that the only information that is used in the computation of the numeric or QR codes are the identity keys and the users' identities. These authentication protocols can only protect against an adversary injecting its own identity keys, but not against an adversary that is capable of injecting signed prekeys or ephemeral keys (whether ratcheting, or one-time prekeys) without detection.

Consider the attack shown in Figure 2. The adversary corrupts the long-term identity key idk_B of the responder, and then uses it to authenticate its own maliciously generated signed prekey and one-time keys. Since the adversary has not injected its own distinct identity key (but instead uses the responder's public key), the fingerprints are computed identically on both sides, and thus the communicating parties do not detect the malicious behaviour of the adversary.

Thus, the Signal authentication protocol only offers a very weak form of entity authentication. One justification for not providing session authentication is deniability. Namely, by not linking message keys to the authentication, messages are deniable. This comes at the cost of no authentication following compromise events, despite the complexity of Signal aiming to address such a threat model.

One-way QR code authentication. QR code authentication in Signal functions by one device scanning the QR code displayed by the second device. A confirmation notice is displayed to the user on the first device if the QR code is correct. No interaction or confirmation is performed on the second device, or is the any device-to-device confirmation of the result of the QR code scan. The one-way nature of the QR code matching implies that the user's decision regarding whether or not the entity authentication is successful depends entirely on its interaction with one device. If an adversary was able to interfere with the device's display, such as by a display overlay via malware operating on the one device, then the user could be tricked into approving the authentication.

Note that this differs intrinsically in the trust model from the numeric code comparison alternative. When a user visually compares numeric codes displayed on two devices, they are performing a two-way verification. An attacker would need access to both displays or minimally access to one and full knowledge of the other to trick the user in the above sense. For QR code verification, the attacker requires only access to the display on one device – no knowledge of the display on the second device or any pairing keys is required.

Public code authentication. In the current Signal authentication protocol, all values contained within the code (whether numeric or QR-based) are public. If an adversary is able to interfere with a device's display, such as by a display overlay via malware operating on the one device, then the user could be tricked into approving the authentication. Specifically, if the attacker can control a user's display, they can perform MitM attacks by injecting public keys, and simply compute the expected code for the user's display.

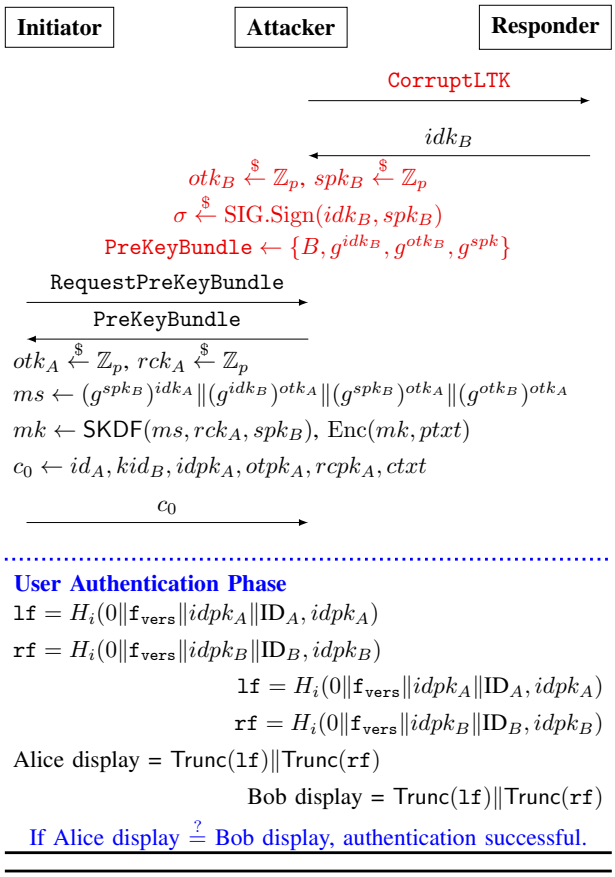

Figure 2. An attack on session authentication against users using authentication mechanisms. Note that kid_B denotes both the identifiers for Bob's one-time prekey and Bob's signed prekey, for conciseness. SKDF denotes the iteration of KDF steps necessary to compute a message key mk from the initial secrets used in the X3DH protocol, and H_i is an iterated hash function.

Weak User Mediation. Users play an active role in the entity/device authentication process, whether by comparing numeric codes or scanning QR codes. However, the Signal authentication protocol is even more reliant on the user, allowing the user to *decide* the final authentication outcome. Namely, after comparison, a user may choose to select "Verified" or not. The user may also later deselect the "Verified" status of the session. It should be noted that marking a session "Verified" does not in fact require any comparison to take place, even for comparison of QR codes. Consequently, it is possible for the malicious third party E to trick a user A into believing that A's session with B is verified when it has not been. This is possible via social engineering or temporary access to the device.[3]

Notably, a user may continue a session even after a comparison failure, and without *any* warning in the session. This is most relevant for QR code comparison, where an automatically issued warning to the user is possible. Even after comparison failure, the user may mark the conversation as verified.

3. Note that such access does necessarily imply access to the message contents of the session.

3. Modified Signal Authentication

In this section we describe our *Modified Device-to-User Signal Authentication* (MoDUSA) protocol. Recall that achieving APCS healing for entity authentication requires session-specific information to be tied to verification checks, as we discuss in Section 2.4. One possible, but naive, approach to this would be to include all public keys exchanged during the protocol execution in the verification check. However, an attacker in control of the user display could generate the same fingerprints as a valid user and post it for comparison elsewhere, since it is generated over public information, as described in Section 2.4. It is therefore necessary to tie together proof of private key ownership over all ratchets. This leads to our MoDUSA construction, where we compute MACs using PCS secrets over session-specific information. Recall that achieving APCS healing for entity authentication requires session-specific information to be tied to verification checks. One possible, but naive, approach to this would be to include all public keys exchanged during the protocol execution in the verification check. However, an attacker in control of the user display could generate the same fingerprints as a valid user and post it for comparison elsewhere, since it is generated over public information. It is therefore necessary to tie together proof of private key ownership over all ratchets. Naturally this change addresses the issues described under both No Session Authentication and Public Code Authentication in 2.4. This leads to our MoDUSA construction, where we compute MACs using PCS secrets over session-specific information.

MoDUSA provides epoch-level authentication and is strong against both device and user compromise attacks. Similarly to the Signal authentication protocol described in Section 2.3, MoDUSA is intended to work using both scannable QR codes and displayable numeric codes, offering different levels of user interaction and security (as in Table 1). In both cases we must first expand the Double Ratchet protocol's key schedule.

In Figure 3 we describe our modification to the key schedule for each asymmetric ratchet, which adds the derivation of an additional authentication key ak^i for each epoch i (associated with an asymmetric ratchet)[4]. The key will be used in generating the fingerprints that will be displayed (in the numeric code variant), or scanned (in the QR code variant).

One difficulty in creating an authentication protocol that will iteratively cover all epochs is that Signal is an asynchronous messaging protocol, run over a potentially lossy channel. Thus, it is impossible to guarantee that any single message from a message chain has been delivered from sender to receiver. As a result, our MoDUSA protocol (or any authentication protocol) cannot be used to guarantee user agreement on all messages sent and received by the devices. However, due to the strict "ping-pong" nature of the asymmetric ratchet used in Signal's Double Ratchet protocol, it is possible to guarantee that at least *one* message of the previous epoch from the communicating partner has been received, as otherwise the partner does not generate a new ratchet key, and thus the

4. Coincidentally, the implementation we examined already derives a third unused key for each epoch [25].

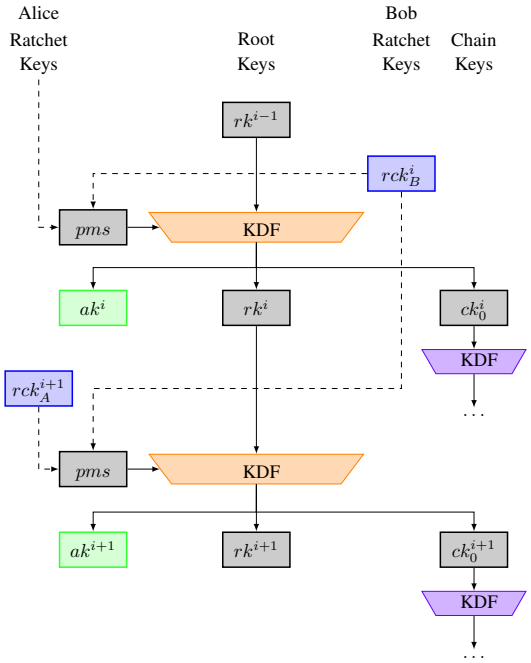

Figure 3. The modified key schedule for the Signal Double Ratchet protocol. We expand the output of the KDF for each ratchet, outputting ak^i for each "epoch" i as well as the standard Signal outputs (rk^i, and ck_0^i).

epoch does not increment forward. It follows that since a party will continue using the same ratchet key until a new ratchet key arrives from their partner, the view of ratchet public keys from both parties will match, except that one view *may* have one additional ratchet key, and further desynchronization (without malicious behavior) cannot occur. Consequently, for correctness in MoDUSA we have that devices must agree either on:

- all ratchet keys sent between the two devices, or
- all ratchet keys sent between the two devices, except for the last ratchet key,

If a full chain of messages has been dropped (or not yet delivered), one device would believe that it is in epoch i, while the other believes that it is in epoch $i-1$.

As such, to compute fingerprints in the MoDUSA protocol, each device maintains and updates two chaining hash values with each asymmetric ratchet[5]:
$H^{i-1} = \mathrm{H}(\texttt{PKB} \| idpk_A \| idpk_B \| otpk_A \| rcpk_A^0 \| \ldots \| rcpk_A^{i-1})$ and $H^i = \mathrm{H}(\texttt{PKB} \| \ldots \| rcpk_A^{i-1} \| rcpk_B^i)$, where PKB is the prekey bundle received by the initiator, and $H^0 = \mathrm{H}(\texttt{PKB} \| idpk_A \| idpk_B \| otpk_A \| rcpk_A^0)$.

Note that this contains all the public keyshare and identity information used in the initial X3DH key exchange, in addition to all ratchet keys used in each asymmetric ratchet up to the point of fingerprint computation. Since all hash input information is public, no secret keys are maintained in memory for this computation, and consequently this does not affect forward secrecy.

In addition to the hash chains, each party maintains ak^{i-1}, ak^i, the two most recently computed authentication keys (see Figure 3). However, we distinguish the computation of fingerprints for QR or numeric codes. The reason for this is that *META*-CompUser security (including adversarial read ability on the UtD channel) is achievable under QR codes using role separation as discussed in Section 5. Meanwhile single fingerprint numeric comparison does not allow for role separation. More details appear under the analysis of the MoDUSA protocol in Section 5.1. For displayable numeric codes, fingerprints are computed as $\texttt{fprint}^i = \mathsf{HMAC}(ak^i, H^i \| \texttt{fvers})$.

For QR codes, fingerprints are computed as $\texttt{fprint}^i = \mathsf{HMAC}(ak^i, H^i \| \texttt{fvers} \| role)$ where $role$ is the role of the device displaying the QR code (instead of scanning). This change causes the QR codes to be domain-separated, and thus even if an attacker has access to one QR code (perhaps via shoulder-surfing), the attacker (without access to the underlying secret state) cannot compute (and thus display) the other.

Whenever a device sends or receives an asymmetric ratchet key, a new "epoch" is triggered due to the computation of a new epoch key (which we set as pms, the DH output from each successive pair of ratchet keys). Each party then rolls the chaining hash values forward (i.e $H^{i-1} \leftarrow H^i$, $ak^{i-1} \leftarrow ak^i$, etc.) and computes a new chaining hash value H^i, a new authentication key ak^i and a new fingerprint $\texttt{fprint}^i$. When users compare fingerprints, they compare the highest fingerprint value i that both devices share. If the two fingerprints are equal, then authentication was successful for all epochs, from the initial key exchange (epoch 0) to the most recent shared epoch (epoch $i-1$ or i).

Note that this solution does not imply evolving QR codes, or that fingerprints *need* to be generated per epoch. It is only required that the authentication keys ak evolve per epoch. When the user decides to authenticate, the current authentication keys ak^i and ak^{i-1} can be "locked" (e.g. once the user selects "view safety number" in Signal, ak^i and ak^{i-1} are not changed or overwritten until the user exits that view). The QR code and/or numeric codes can then be computed for the given epoch. More details on usability of this solution are discussed in Section B.

Remark 1. It is natural to ask "why not simply maintain epoch $i-1$ instead of both epochs?" The first reason is that users are not aware if they are ahead or behind their communicating partner, posing the same problems as simply maintaining epoch i. The second reason is that memory management is not significantly reduced by maintaining a single epoch $i-1$. Since future authentication keys ak^i are derived from the previous root key rk^{i-1} and the DH output from the most recent pair of ratchet keys $rcpk^{i-2}$, $rcpk^{i-1}$, the secret values associated with these must be stored to derive the next ak^i regardless.

Arguably the most common practical scenario here is that both parties will have received the most recently generated ratchet key, and thus both parties will agree on the most recent epoch i. Since authentication will occur intermittently in real-world scenarios, we should attempt to authenticate the highest epoch that is possible. However, we consider an asynchronous messaging protocol with a potential lossy channel, and thus must account for this in our authentication solution by allowing authentication at epoch $i-1$.

5. For implementations, these may be computed using iterated hashes such as $H^i = H(H^{i-1} \| rcpk_B^i)$. For simplicity in the security proof we consider a single hash computation.

Remark 2. It can be argued that since the authentication keys ak^i are computed via Diffie–Hellman operations using the public keys (both ratchet and long-term), that it is not necessary to include the chaining hash value in the computation of the fingerprints. This is valid, but the explicit inclusion of the chaining hash value: (a) closely matches the construction of the current Signal fingerprint computation (i.e. computing a hash over public keys), which reduces engineering work required to transition to our proposed solution; and (b) simplifies the proof, as hash collision resistance allows us to straightforwardly argue for security of MoDUSA as long as the channel between the user and the device is secure. This is in comparison to a more involved argument about agreement on the authentication key implying agreement on all public keys exchanged during protocol execution.

4. Authentication Model

In this section we introduce a framework to analyse entity authentication in ratcheted messaging protocols. We formalise entity authentication protocols in the ratcheted key exchange setting, and to prove such protocols secure, we expand the 3-PUMA (3-Party Possession User Mediated Authentication) model [15], which separates out the device-to-device channel and user-to-device channel (e.g. device display and user input). Our 3-PUMA variant, which we call the "Mediated Epoch Three-party Authentication" ($META$) security model, aims at explicit entity authentication and builds on multi-stage security [8], [12] via ratcheted authentication.

User-Mediated Entity Authentication On a high level, user-mediated entity authentication protocols creates an easily-exchanged digest of information, which we denote *fingerprints*.[6] Users then exchange these fingerprints in an out-of-band channel, and decide to accept or reject the authentication attempt.

Recall that the Signal protocol relies on user-mediated entity authentication to establish trust in long-term keys and long-term identifying information. However, this is the limit of its guarantee: the Signal authentication protocol does not authenticate any other information. In particular, it does not authenticate per ratchet update information, and thus users gain no security benefit if the attacker has already exposed the long-term key of either party ($idpk_A$, $idpk_B$).

6. The use of "fingerprint" here is rooted in the Signal terminology, which we stay consistent with for the rest of the paper.

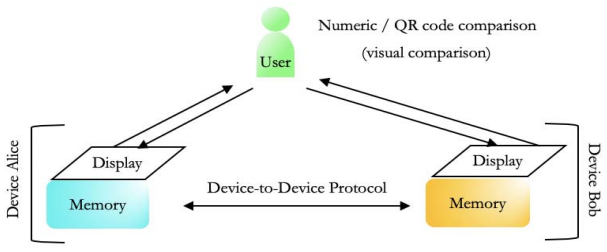

Figure 4. A high-level figure of our setting. The protocol runs on the device, and provides fingerprints to the users. To simplify analysis, in our security model Alice and Bob are considered a single user, with a communication channel existing between Device A and the User, Device B and the User, and between the two Devices.

META **Security Model** We now formalise the setting of our $META$ security model capturing user-mediated authentication protocols; as the name suggests, $META$ extends beyond capturing devices executing such protocols, but also explicitly captures users interacting with the devices within the security model. As a result, $META$ can capture a broader class of attackers: that compromise secret state within the device, and also compromise the channel between the device and the users.

In addition, $META$ is primarily concerned with asynchronous messaging protocols, and this is reflected in how we capture authentication. An asynchronous protocol is one that does not require both parties to be online at the same time. As a result of this restriction, in our setting we do not have a guarantee that both parties have received all messages sent between the communicating devices. One of the technical difficulties our model addresses then is determining the right "level" of authentication: due to our focus on *ratcheted* protocols, we decide to target entity authentication *per asymmetric ratchet*, which we refer to as *epochs*. As in [15], we use session identifiers instead of matching conversations for defining partnering, which is more standard in authenticated key exchange models. This is due to the expected asynchronicity between the three communication channels (User-to-DeviceA, DeviceA-to-DeviceB, and User-to-DeviceB), which precludes matching conversations. Ratcheting protocols are a form of continuous key agreement and consequently have an evolving session identifier. We indicate the session identifier at any given epoch as `epid` and call it an *epoch identifier*.

Note that to simplify analysis and complexity in our framework, we follow the direction of [15], and model the two device users as a single user, see Figure 4.

Protocol Participants. A participant in a $META$ protocol is either a device $I \in \mathcal{ID}$ or a user U. The set of all participants is the union $\mathcal{ID} \cup \{U\}$, where the elements of $\mathcal{ID}$ are *devices* or *identities*. There may be multiple *sessions* at any participant, such that π_s^P is the s-th session at P. Below is a description of the internal state of the two types of participants. At the beginning of the experiment, the challenger $\mathcal{C}$ generates a list of n_P public keys pairs for each device $I \in \mathcal{ID}$, $(sk_1, pk_1), \ldots (sk_{n_P}, pk_{n_P})$. Note that devices do not have access to the list of public keys; they instead set partner public keys during the protocol.

To capture the Signal authentication protocol (and eventually our modified version) in $META$, devices maintain values used by these protocols – specifically, long-term public keys (sk, pk), any secret state (i.e., rck), and ratcheting secret outputs $esk[\mathsf{T}]$, which we denote *epoch secret keys*. For example, in the Signal Protocol the epoch secret keys $esk[\mathsf{T}] = rcpk_{\mathsf{T}-1}^{rck_{\mathsf{T}}}$ are derived from the asymmetric ratchets sent between both parties (see Section 2). This allows our model to be specific about the values that are compromised in the $META$ security experiment, simplifying how we capture freshness conditions and in particular, post-compromise security.

Devices In $META$, each device $I \in \mathcal{ID}$ is modelled as a set of session oracles, where each session maintains the following list of variables:

- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$: a variable indicating the role of I in the session.

- $\mathsf{T} \in \mathbb{N} \cup \perp$: A counter indicating the current epoch of the session, initialised as $\perp$.
- $st[\mathsf{T}] \in \{0,1\}^*$: a variable storing any additional state for a given epoch T, initialised as $\perp$.
- $esk[\mathsf{T}] \in \mathcal{ESK}$: a variable storing the private epoch secret key output for a given epoch T, where $\mathcal{ESK}$ is the private epoch key space. Initialised as $\perp$. Updated ratchets to $esk[\mathsf{T}]$ are denoted $esk[\mathsf{T}] \leftarrow esk[\mathsf{T}+1]$ (note that this also implies $\mathsf{T} \leftarrow \mathsf{T}+1$).
- $\mathtt{pid} \in \mathcal{ID} \setminus \{I\}$: a variable storing the partner identity for the session, initialised as $\perp$.
- $pk_{\mathtt{pid}} \in \mathcal{PK}$: a variable storing the public key for the session partner, where $\mathcal{PK}$ is the public key space, initialised as $\perp$. This variable is set during the protocol execution.
- $(sk, pk) \in \mathcal{SK} \times \mathcal{PK}$: a variable storing the private and public key for I, where $\mathcal{SK} \times \mathcal{PK}$ is the private/public key space.
- $\alpha[\mathsf{T}] \in \{\mathsf{accept}, \mathsf{reject}, \perp\}$: a variable indicating if the session accepts for a given epoch T, rejects, or has not yet reached a decision. Initialised as $\perp$.
- $\mathtt{epid}[\mathsf{T}] \in \{0,1\}^* \cup \perp$: a variable storing the epoch identifier at each epoch T, initialised as $\perp$.

The internal state of each session oracle π_s^I owned by identity I is initialized to $(\mathsf{role}, \mathsf{T}, st[\mathsf{T}], esk[\mathsf{T}], \mathtt{pid}, pk_{\mathtt{pid}}, (sk, pk), \alpha[\mathsf{T}], \mathtt{epid}[\mathsf{T}]) = (\perp, \perp, \perp, \perp, \perp, \perp, (sk_I, pk_I), \perp, \perp)$. We disallow $\mathtt{pid}_I = I$, as devices do not authenticate themselves.

Users U is similarly modelled via session oracles, where each session π_t^U maintains at minimum the following variables:

- Two device-session pair identifiers (I, s) and (I', s').

This follows practice that a user should identify messaging conversations that it wishes to authenticate.

Definition 1 (Matching Epoch ID). We say that identities I and I' have *matching epoch IDs* for sessions s and s', respectively, and for an epoch T if $\pi_s^I.\mathtt{epid}[\mathsf{T}] = \pi_{s'}^{I'}.\mathtt{epid}[\mathsf{T}]$, where $\mathsf{T} \neq \perp$ and $\pi_s^I.\mathtt{epid}[\mathsf{T}] \neq \perp$.

Note that unlike most key exchange or authentication models, we do not require prefix-matching. This is an artifact of asynchronous messaging protocols using lossy channels. Each epoch represents a chain or flow of messages from one device to another, rather than individual messages themselves, as we have no guarantees that any given message in the flow reaches the destination. Thus our epoch identifiers are initialised as $\emptyset$, updated only once, and we need to consider only exact matching.

Definition 2 (Partnering Device to Device). Two sessions $\pi_s^I, \pi_{s'}^{I'}$, with $I, I' \in \mathcal{ID}$, are *partnered* in an epoch T if $\pi_s^I.\mathtt{pid} = I'$, $\pi_{s'}^{I'}.\mathtt{pid} = I$, $\pi_s^I.\mathsf{role} \neq \pi_{s'}^{I'}.\mathsf{role}$, $\pi_s^I.\alpha[\mathsf{T}] = \pi_{s'}^{I'}.\alpha[\mathsf{T}] = \mathsf{accept}$, and finally, $\pi_s^I.\mathtt{epid}[\mathsf{T}]_{s,I} = \pi_{s'}^{I'}.\mathtt{epid}[\mathsf{T}]$.

Let $\mathcal{A}$ be a probabilistic polynomial-time (PPT) algorithm against authentication with the following abilities and allowed queries in the experiment $\mathsf{Exp}^{META\text{-}\mathsf{type},\mathcal{A}}_{\Pi,n_P,n_S,n_T}$.

We highlight that there are two communication channels that are modelled in our *META* security framework. The Device-to-Device channel, captures a "standard" network modelled in most cryptographic protocols, and models messages sent between devices, perhaps over the internet. The second, User-to-Device, captures the channel

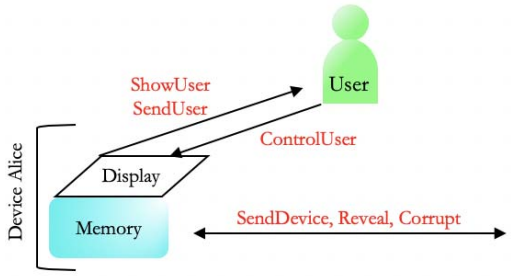

Figure 5. A high-level view of allowed *META* queries. Adversarial control on the depicted channels is enabled through the associated queries.

between a User and a Device. This captures messages displayed on the screen of a smart phone, and the keyboard that a User enters input to the device over.

Device-to-Device (DtD) The adversary is able to read, modify, replay, reorder, and delete messages between participants I and I', such that $I, I' \in \mathcal{ID}$.

User-to-Device (UtD) The adversary may not modify a message's sender/recipient for messages between identities $I \in \mathcal{ID}$ and the user U. The adversary is allowed to read, replay, reorder, and delete UtD messages, but may not modify UtD messages.

The distinction between the DtD channel and the UtD channel directions can be seen in Figure 5.

The above adversarial abilities are standard such that the adversary can control the network (DtD). Adversarial message modification is restricted between the user and device only (UtD), thereby capturing the concept that the adversary cannot modify what the user sees or inputs on the device. However, we provide a query that allows complete compromise of the user whereby the adversary can gain such control. The reason for this is similar to allowing protocol participant corruption in the traditional sense. A strong protocol should be robust to user compromise (i.e. such as those protocols not relying on the user at all), such that an adversary cannot falsely force authentication in spite of controlling user activity. Meanwhile, a weak protocol depends entirely on an honest and reliable user. This aligns with the historical development of key exchange models, where security could not meaningfully be considered under a party's corruption for weak protocols, but with advent of better understanding of protocol security models, such compromise becomes possible for some protocols which do not rely solely on long-lived keys. We expect that user control, or conditional user control, need not be fatal to security for all user-mediated authentication protocols.

Queries The adversary is able to interact with the Devices and Users with the following queries:

- $\mathsf{SendDevice}(\pi_s^I, m)$. The adversary sends a message m to a session oracle π_s^I. The message is processed according to the protocol specification and any response is returned to the adversary. If π_s^I (where $I \in \mathcal{ID}$) receives m as a first message, then the oracle checks if m consists of a special initiation message $(m = (\mathsf{init}, I'))$, for $I' \in \mathcal{ID}$, to which it responds by setting $\mathtt{pid} = I'$, $\mathsf{role} = \mathsf{initiator}$, and outputs the first protocol message. Otherwise it responds by setting $\mathtt{pid} = I'$, $\mathsf{role} = \mathsf{responder}$, and responding according to the protocol specification.
- $\mathsf{SendUser}(\pi_t^U, m)$. Using this query, the adversary sends a message m to a session oracle of his choice,

where π_t^U is an oracle for session t at user U. The message is processed according to the protocol specification and any response is returned to the adversary. If $\nexists I, s$ such that m has been honestly generated by π_s^I, this query outputs $\perp$.
If a session oracle π_s^U receives m as a first message, then the oracle checks if m consists of a special initiation message $m = (\mathsf{init}, (I, I'))$, for $I, I' \in \mathcal{ID}$, to which it responds by setting $\mathsf{initiator} = I$ and $\mathsf{responder} = I'$. Otherwise it outputs $\perp$.

- $\mathsf{Reveal}(\pi_s^I, \mathsf{T})$. This query returns the epoch key $esk[\mathsf{T}]_{s,I}$ as well as any additional epoch session state $st[\mathsf{T}]$ of the T-th epoch for the s-th session for the identity $I \in \mathcal{ID}$.
- $\mathsf{Corrupt}(I)$. This query returns the private key sk_I of identity $I \in \mathcal{ID}$.
- $\mathsf{ShowUser}(I)$. This query returns $\perp$. After this query the adversary is allowed to modify or create any UtD message from I to the user.
- $\mathsf{ControlUser}()$. This query returns $\perp$. After this query the adversary is allowed to modify or create any UtD message from U.
- $\mathsf{Test}(\pi_s^I)$: This query initiates user interaction with the devices, in the current epoch at π_s^I, and according to protocol specification. The query returns the result of the protocol execution.

We give a diagram of the channels between Devices and the User in Figure 5.

We separate out an adversary's ability to control user input to devices and display device output to a user via ControlUser and ShowUser queries. ControlUser models an adversary's ability to take full control of the user (such as by acting as the user itself or through social engineering). It provides the adversary with the ability to modify and inject messages from the user to the device. In contrast, ShowUser provides the adversary with the ability to modify and inject messages from the device to the user, i.e. allowing an adversary to control the display and manipulate what a user sees. For example, malware on a device may manipulate the output interface, despite not gaining access to any actual secret values.

As in traditional key exchange models, Corrupt compromises the long-term key of the device, and Reveal compromises all other state of the device at a particular epoch. Consequently, ShowUser is meaningfully distinct from a Reveal or Corrupt query which apply strictly to the device secrets. Finally, SendUser allows the adversary to display honestly generated messages from the device to the user on the User-to-Device channel and SendDevice allows the adversary to send both honest and malicious messages between any two devices. Various query combinations may be used to capture different attacks.

Consider the One-way QR Code Authentication issues in Section 2.4: our model captures this through the following queries. The adversary issues a ShowUser query on one device, gaining display capabilities (e.g. through malware). Through use of the public code authentication in Signal, the adversary may pre-compute and now display the expected QR code (or numeric value) to the user, without issuing a Reveal or Corrupt query. Alternatively, consider how these queries model the Tap n' Ghost pairing attack: the adversary issues a ShowUser query to gain display control, prompting the user to deny the pairing request.

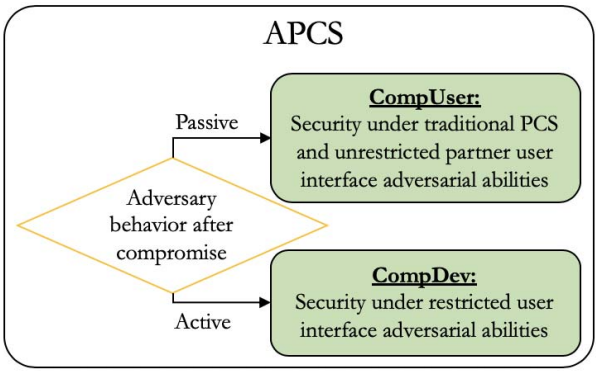

Figure 6. APCS security connection to PCS, CompUser, and CompDev security.

The adversary then calls ControlUser, correlating to the touch-screen adaptive ploy, so that the user's denial of the pairing request is recorded as approval and the pairing completes. Modelling of such attacks is not possible using solely traditional Reveal and Corrupt queries.

At this point, we describe freshness conditions for our security experiment. On a high-level, freshness conditions (which we separate into *device* and *user* freshness) restrict the adversary from issuing Reveal, Corrupt or ShowUser and ControlUser (for *device* and *user* compromises, respectively) queries that would allow them to trivially win the security experiment.

We begin by introducing the first threat setting, compromised user (CompUser), where an adversary is allowed to inject messages displayed to the user U from one of the devices in an epoch T, but is restricted from being able to trivially expose secrets associated with that epoch. This can essentially be seen as equivalent to previous notions of PCS, where there exists no OOB channel, and the adversary must be passive in the device-to-device channel at some point. Even after an adversary learns the keys associated with a session, if a fresh update to the keying material is sent and received while the adversary is passive, it should not be possible for the adversary to force a device into accepting an authentication attempt without actual matching epoch identifiers. Dependent on the choice of epoch identifier, satisfaction of CompUser security may imply authentication of not only the current epoch, but the entire session up to that epoch (e.g. see Section 3).

Secondly, we introduce the compromised device (CompDev) setting, where an adversary is allowed to expose secrets at will, but is restricted from (separately) controlling what is displayed to the user and the user input back to the device. This completes our APCS extension, where active attackers modifying messages on the DtD channel are not required to be passive, but instead can be detected via entity authentication, as long as the attacker cannot modify messages between the user and the target device. Fig. 6 shows the relationship between APCS achieved by the *META* model, CompUser and CompDev security, and traditional PCS.

Definition 3 (Device Freshness CompUser). An oracle π_s^I for an identity $I \in \mathcal{ID}$ is called *fresh under compromised user (CompUser-fresh)* for an epoch T if the following hold:

1) If $\exists \mathsf{T}^*$, such that $\mathcal{A}$ issued a query $\mathsf{Reveal}(\pi_s^I, \mathsf{T}^*)$ (resp. $\mathsf{Reveal}(\pi_{s'}^{I'}, \mathsf{T}^*)$) then

- $\mathtt{pid}_{I,s} = I'$ and $\exists s'$ such that $\mathtt{pid}_{I',s'} = I$, and
- $\pi_s^I.\mathsf{role} = \mathsf{initiator}$ and T is even (resp. odd), or $\pi_s^I.\mathsf{role} = \mathsf{responder}$ and T is odd (resp. even), and
- $\exists \mathsf{T}', \mathsf{T}''$, where $\mathsf{T} \geq \mathsf{T}' > \mathsf{T}'' > \mathsf{T}^*$ such that $esk[\mathsf{T}']_{s,I} = esk[\mathsf{T}']_{s',I'}$ and $esk[\mathsf{T}'']_{s,I} = esk[\mathsf{T}'']_{s',I'}$ and $\mathcal{A}$ has not issued $\mathsf{Reveal}(\pi_s^I, \bar{\mathsf{T}})$ or $\mathsf{Reveal}(\pi_{s'}^{I'}, \bar{\mathsf{T}})$ where $\bar{\mathsf{T}} \in \{\mathsf{T}', \mathsf{T}''\}$.

2) If $\exists \mathsf{T}^*$, such that $\mathcal{A}$ issued a query $\mathsf{Reveal}(\pi_s^I, \mathsf{T}^*)$ (resp. $\mathsf{Reveal}(\pi_{s'}^{I'}, \mathsf{T}^*)$) then
   - $\mathtt{pid}_{I,s} = I'$ and $\exists s'$ such that $\mathtt{pid}_{I',s'} = I$, and
   - $\pi_s^I.\mathsf{role} = \mathsf{initiator}$ and T is odd (resp. even), or $\pi_s^I.\mathsf{role} = \mathsf{responder}$ and T is even (resp. odd), and
   - $\exists \mathsf{T}'$, where $\mathsf{T} \geq \mathsf{T}' > \mathsf{T}^*$ such that $esk[\mathsf{T}']_{s,I} = esk[\mathsf{T}']_{s',I'}$, and $\mathcal{A}$ has not issued $\mathsf{Reveal}(\pi_s^I, \mathsf{T}')$ or $\mathsf{Reveal}(\pi_{s'}^{I'}, \mathsf{T}')$.
3) If $\mathcal{A}$ issued a $\mathsf{Corrupt}(I)$ or $\mathsf{Corrupt}(I')$ query at any time, where $\mathtt{pid}_{I,s} = I'$, then $\exists s', \mathsf{T}'$ such that $\mathtt{pid}_{I',s'} = I$, $\mathsf{T} \geq \mathsf{T}'$, and $esk[\mathsf{T}']_{s,I} = esk[\mathsf{T}']_{s',I'}$.

Definition 4 (User Freshness CompUser). A user U is called *fresh under compromised user (CompUser-fresh)* for an epoch T at a session oracle π_s^I, $I \in \mathcal{ID}$, unless

- $\mathsf{ShowUser}(I)$ is issued before or during epoch T in π_s^I, or
- $\mathsf{ControlUser}()$ is issued before or during epoch T in π_s^I.

On a high-level, Definitions 3 and 4 restrict the adversary from compromising session secrets, preventing the adversary from trivially computing the expected authentication code even if they have compromised the UtD channel between the partner and the user via ShowUser queries. Since epoch level authentication is dependent on both long-term and epoch-specific keys, we allow corruption of long-term keys after such a passive epoch in Definition 3. Note that this applies to the CompUser threat model – under CompDev below, we capture the fully active adversary on the DtD channel.

Definition 5 (Device Freshness CompDev). A session oracle π_s^I for an identity $I \in \mathcal{ID}$ is always called *fresh under compromised device (CompDev-fresh)* for an epoch T, regardless of Corrupt and Reveal queries issued by the adversary.

Definition 6 (User Freshness CompDev). A user U is called *fresh under compromised device (CompDev-fresh)* for an epoch T at a session oracle π_s^I, $I \in \mathcal{ID}$, unless

- $\mathsf{ShowUser}(I)$ is issued before or during epoch T in π_s^I, or
- $\mathsf{ShowUser}(I')$ is issued before or during epoch T in $\pi_{s'}^{I'}$, where $\pi_{s'}^{I'}$ and π_s^I are partnered in T, or
- $\mathsf{ControlUser}()$ is issued before or during epoch T in π_s^I.

We define device freshness under device compromise as syntactic sugar for the following experiment. Particularly, we consider security under *META-CompUser* and *META-CompDev*, with a combined *META* experiment dependent on the device and user freshness combination selected (*CompUser* or *CompDev*).

Definition 7 (META Experiment). Let $\mathcal{A}$ be a PPT adversarial algorithm against a user mediated authentication protocol Π, interacting with a challenger via the queries defined above in the experiment $\mathsf{Exp}^{META-\mathsf{type},\mathcal{A}}_{\Pi,n_P,n_S,n_T}$, where n_T is the maximum number of epochs, n_P is the maximum number of devices, n_S is the maximum number of sessions at any party, and type is a freshness type. We say that the challenger outputs 1, denoted $\mathsf{Exp}^{META-\mathsf{type},\mathcal{A}}_{\Pi,n_P,n_S,n_T}(\lambda) = 1$, if a Test query is made in session π_s^I and any of the following conditions hold at any $\mathsf{T} \neq \perp$:

1) *Matching* epid*, but no acceptance.*
   - *META*-type is CompDev and
   - Oracles π_s^I and $\pi_{s'}^{I'}$ have matching epid and
   - either π_s^I or $\pi_{s'}^{I'}$ does not accept and
   - the user is type-fresh in T at π_s^I.
2) *Acceptance, but no matching* epid.
   - There exists a type-fresh oracle π_s^I at epoch T which has accepted and
   - there is no partner oracle $\pi_{s'}^{I'}$ at epoch T which is type-fresh and
   - the user is type-fresh in T at π_s^I.

Otherwise the experiment outputs 0.

We define the advantage of the adversary $\mathcal{A}$ in the experiment $\mathsf{Exp}^{META-\mathsf{type},\mathcal{A}}_{\Pi,n_P,n_S,n_T}(\lambda)$ as

$$\mathbf{Adv}^{META-\mathsf{type},\mathcal{A}}_{\Pi,n_P,n_S,n_T}(\lambda) := \Pr[\mathsf{Exp}^{META-\mathsf{type},\mathcal{A}}_{\Pi,n_P,n_S,n_T}(\lambda) = 1] \quad .$$

Definition 8 (Security of META-CompUser). We say that a user mediated protocol Π is *META-CompUser secure* if there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all PPT adversaries $\mathcal{A}$ interacting according to the experiment $\mathsf{Exp}^{META-\mathsf{CompUser},\mathcal{A}}_{\Pi,n_P,n_S,n_T}(\lambda)$, it holds that

$$\mathbf{Adv}^{META-\mathsf{CompUser},\mathcal{A}}_{\Pi,n_P,n_S,n_T}(\lambda) \leq \mathsf{negl}(\lambda) \quad .$$

Definition 9 (Security of META-CompDev). We say that a user mediated protocol Π is *META-CompDev secure* if there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all PPT adversaries $\mathcal{A}$ interacting according to the experiment $\mathsf{Exp}^{META-\mathsf{CompDev},\mathcal{A}}_{\Pi,n_P,n_S,n_T}(\lambda)$, it holds that

$$\mathbf{Adv}^{META-\mathsf{CompDev},\mathcal{A}}_{\Pi,n_P,n_S,n_T}(\lambda) \leq \mathsf{negl}(\lambda) \quad .$$

Note that under *META*-CompUser, it is not possible to require that matching epid implies acceptance (and hence is excluded under the first condition of Def. 7). Under this model type, the best that can be required is that acceptance at two devices implies possession of a matching epid. Protocols that rely on user mediation cannot force acceptance if the information provided to the user (e.g. through a device display – ShowUser query) is corrupted.

5. Security Analysis

The Signal authentication protocol does not provide *META-CompUser* or *META-CompDev* security. Here we provide one counter-example for each.

META-CompUser Signal fails to meet *META-CompUser* due to the unilateral nature of QR codes. A ShowUser query can be made on the device scanning the QR code, enabling failure of condition 2) of Def. 7 despite user freshness (the attacker is able to display a simple confirmation to the user, despite a non-match). Note that

I	I'	**CompDev w/o E.**	**CompDev w/ E.**	**CompUser w/o E.**	**CompUser w/ E.**
Display match	Display match	✓	✓	✓	X
Display match	Scan match	✓	✓	X	X
Scan match	Display match	✓	✓	✓	X
Scan match	Scan match	✓	✓	✓	X
Display no match	Scan no match	✓	✓	X	X
Scan no match	Display no match	✓	✓	✓	✓
Scan no match	Scan no match	✓	✓	✓	✓

TABLE 1. CROSS COMPARISON OF *achievable* SECURITY GUARANTEES BASED ON A MoDUSA SESSION IDENTIFIER, WHERE **DISPLAY** REFERS TO A STATIC VERIFICATION CODE DISPLAY, **SCAN** REFERS TO A YES/NO CONFIRMATION ON THE OTHER DEVICE'S STATIC DISPLAY (E.G. QR CODE SCANNER), **MATCH** REFERS TO MATCHING VERIFICATION CODES ON BOTH DEVICES, AND **NON-MATCH** REFERS TO SEPARATE VERIFICATION CODES FOR AUTHENTICATING INITIATOR I AND AUTHENTICATING RESPONDER I'. **E.** IS AN EAVESDROPPER ALIGNED WITH THE ADVERSARIES ABILITY TO READ UTD MESSAGES. **CompUser** AND **CompDev** (DEFINED IN DETAIL IN SECTION 4) REFER TO THE CONDITIONS OF A COMPROMISED USER (E.G. ADVERSARIAL CONTROL OF UTD CHANNEL) AND COMPROMISED DEVICE (E.G. ADVERSARIAL CONTROL OF DTD CHANNEL), RESPECTIVELY.

this attack also easily holds if a ShowUser query is made on the device displaying the QR code instead, since the QR code relies only on long-term static public information.

META-CompDev

Signal authentication protocol fails to meet *META-CompDev* under condition 2) of Definition 7. Suppose $\mathcal{A}$ issues a Reveal(I) query and a Reveal(I') query at epoch T and epoch $\mathsf{T}+1$, respectively. These queries return the full epoch state rk^i, allowing $\mathcal{A}$ to fully MitM the communication by providing ratchet key shares of its choice to I and I', respectively. Since the Signal authentication protocol uses only on the long-term keys, an active MitM session-specific modification is successful. More details of similar weaknesses appear in Section C.

Table 1 shows a comparison of various achievable CompUser and CompDev scenarios in the context of scannable QR codes (denoted"scan") and displayable numeric code comparison ("display"). Displayed or scanned values can either be the same on both devices when role separation is not considered ("match") or different when the values are computed separately for each role ("no match"). Scan and display categories are usually paired (e.g. QR code scanning a static displayed code) but scan can also be bi-directional, where each device scans the other. *META* explicitly allows eavesdroppers on the UtD channel (the adversary may read messages sent between the device and a user); however, for illustration purposes Table 1 compares CompUser and CompDev both under the presence and absence of an eavesdropper (**E.**). QR codes are considered separately as they allow for distinct codes for each role, with comparison partially reliant on the internal scan computation vs. only displayed information.

In the **CompDev w/o E.** and **CompDev w/ E.** columns of Table 1 we can see the case where the adversary does not have CompUser abilities. When the user acts as a trusted third party over a secure channel, the adversary can only act on the DtD channel and the user may be able to detect a non-match (these are the "typical" OOB analysis scenarios). Under CompUser, the adversary may issue a ShowUser(I') query during the target epoch. As such, for **CompUser w/o E.**, the adversary may show the user a "scan was successful" result, regardless of the actual outcome, if I uses a display only. The same applies to **CompUser w/ E.**, with the addendum that an adversary can also read displays on I and display these to the user on I' after I is put in scan mode. When displays include a role separation and do not match, eavesdropping no longer provides an advantage; placing I in scan mode furthermore means that the adversary cannot succeed against I under CompUser freshness using only a ShowUser(I') query (as can be seen in the final two rows of Table 1). We ignore the case of display non-match / display non-match, as this would imply that the two devices display different numeric codes (computed according to roles), but with no user-decipherable link between the codes.

Notably, to achieve *META*-CompDev security in the presence of an eavesdropper, it is necessary to employ at least one non-legible comparison (e.g. QR codes) with non-matching comparison values for the initiator and responder. This leads us to the MoDUSA protocol of Section 3. MoDUSA is designed with the goal of satisfying *META*-CompUser and *META*-CompDev with use of QR codes, and *META*-CompDev security with use of displayable numeric codes.

5.1. Proofs of MoDUSA Security

Now we set the stage for proving that the proposed MoDUSA protocol achieves *META* security, under both compromised user (CompUser-fresh) and compromised device (CompDev-fresh) attacks. We set the session identifier as the transcript of all public keyshare information (i.e. identity keys, one-time-keys, ratchet keys, and any key identifiers associated with these values) as well as the public identifiers of both parties. This maps naturally to the information used in derivation of the message keys and the fingerprints themselves. This choice of session identifier means that a session identifier for an epoch T is a superstring of the session identifiers for any previous epoch $\mathsf{T}' < \mathsf{T}$. We define the session identifier below.

Definition 10 *(*MoDUSA ***Session Identifier).*** The MoDUSA session identifier at epoch T for a session oracle π_s^I in the *META* experiment is $\pi_s^I.\mathtt{epid}[\mathsf{T}] = \mathtt{PreKeyBundle}\|idk_I\|idk_R\|otk_I\|rcpk_I^0\|\ldots\|rcpk_{\mathsf{role}}^{\mathsf{T}-1}$.[7]

Note that the above selection of session identifier pertains to specifically to ratchet keys and not message keys (mk^{epoch}), as Signal is meant to be robust and handle a large number of dropped messages. As a result, our solution generically does not capture an attacker that injects individual messages in an epoch, but instead attempts to inject *ratchet keys*. If a solution can prevent an adversary from injecting ratchet keys, then they must be able to

7. I and R here are abbreviations for initiator and responder, and role refers to the role of the party that sent the T-th ratchet key. If T is even, then role = initiator, otherwise role = responder.

continuously compromise message keys, thus raising the level of difficulty for an active attacker.

Definition 11 *(*MoDUSA ***Epoch Secret Key).*** The MoDUSA epoch secret key for an epoch T maintained by a session oracle π_s^I in the *META* security experiment is $\pi_s^I.esk[\mathsf{T}]_{=}rcpk_{\overline{\mathsf{role}}}^{\mathsf{T}\text{-}1}rck_{\mathsf{role}}^{\mathsf{T}}$. Note that role and $\overline{\mathsf{role}}$ here refer to the role of the session π_s^I and its communication partner. [8]

We can now begin the analysis of the MoDUSA protocol against CompDev attacks. Since Theorem 1 does not rely on role inclusion, it applies to both MoDUSA QR code and displayable fingerprint comparison.

Theorem 1 *(*MoDUSA ***is*** *META-**CompDev-** **secure).*** Numeric code-based and QR-based MoDUSA is *META*-CompDev-secure irrespective of role inclusion. That is, for any PPT algorithm $\mathcal{A}$ in the *META* security experiment under CompDev freshness conditions, $\mathbf{Adv}^{META\text{-CompDev},\mathcal{A}}_{\text{MoDUSA},n_P,n_S,n_T}(\lambda)$ is negligible under the collision-resistance of the hash function.

Proof. We provide a proof sketch here. Full proof details for Theorem 1 appear in the full version. Since $\mathcal{A}$ cannot modify messages between the User and the devices, CompDev security follows from the collision-resistance of the hash function. Both devices compute the fingerprints $\mathtt{fprint}^T = \mathsf{HMAC}(\widetilde{ak^{\mathsf{T}}} = H^{\mathsf{T}}\|\mathtt{f\_vers}\|role)$ honestly and send the fingerprints to the User. If $\mathcal{A}$ has modified any of the public keys used in the protocol execution, H^{T} will differ for both devices, and thus the User will see two distinct fingerprints, and reject the authentication attempt. □

Theorem 2 *(*MoDUSA ***with roles is CompUser-secure).*** QR-code based MoDUSA is *META*-CompUser-secure against compromised users under *role* inclusion. For any PPT algorithm $\mathcal{A}$ in the *META* security experiment under CompUser freshness conditions, $\mathbf{Adv}^{META\text{-CompUser},\mathcal{A}}_{\text{MoDUSA},n_P,n_S,n_T}(\lambda)$ is negligible under the collision-resistance of the hash function H, the kdf security of key derivation function KDF, the euf-cma security of the MAC algorithm HMAC, and the hardness of the sym-ssPRFODH assumptions.

Proof. Due to space restrictions, we provide a proof sketch. Full details for the proof of Theorem 2 appear in the full version. For each case corresponding to a condition in CompUser security, our analysis is structured as a series of game hops, where we begin with the standard *META* security experiment and iteratively make changes in successive games based on the restrictions placed on the adversary to prevent trivial wins. In the final game, we show that the adversary cannot force a test session (i.e., a session π_s^I such that $\mathcal{A}$ has issued $\mathsf{Test}(\pi_s^I)$) to accept without a matching partner after executing an authentication attempt. For each case the proof is structured almost identically and differences are mostly linked with whether or not the test session is acting as the initiator in the initial key exchange. We give a summary of each game hop below:

G0: This is the standard *META* game, with CompUser freshness conditions. Thus we have $\mathbf{Adv}^{META\text{-CompUser},\mathcal{A},C_1}_{\text{MoDUSA},n_P,n_S,n_T}(\lambda) = \mathbf{Adv}(break_0)$.

G1: In this game abort if there is ever a hash collision. Thus we have $\mathbf{Adv}(break_0) \leq \mathbf{Adv}(break_1) + \mathbf{Adv}^{\mathsf{coll},\mathcal{B}_2}_{H}(\lambda)$.

G2: In this game we guess the index of the first session π_s^I that reaches a status $\pi_s^I.\alpha[\mathsf{T}] \leftarrow \mathsf{accept}$, and abort if there is no session $\pi_{s'}^{I'}$ such that $\pi_{s'}^{I'}.\mathtt{epid}[\mathsf{T}] = \pi_s^I.\mathtt{epid}[\mathsf{T}]$. Thus we have $\mathbf{Adv}(break_1) \leq n_P \cdot n_S \cdot \mathbf{Adv}(break_2)$.

G3: In this game we guess the index of the intended session partner $\pi_{s'}^{I'}$ such that $\pi_s^I.\mathsf{pid} \leftarrow I'$, and $\pi_{s'}^{I'}.esk[\mathsf{T}''] = \pi_s^I.esk[\mathsf{T}'']$, and abort if there is no such session $\pi_{s'}^{I'}$. Thus we have $\mathbf{Adv}(break_1) \leq n_S \cdot \mathbf{Adv}(break_2)$.

G4: In this game we guess the index of the epoch T'' such that $\pi_s^I.esk[\mathsf{T}''] = \pi_{s'}^{I'}.esk[\mathsf{T}'']$, and $\mathsf{T}'' > \mathsf{T}^*$, and abort if we guess incorrectly. By the CompUser definition, this epoch must exist. Thus we have $\mathbf{Adv}(break_3) \leq n_T \cdot \mathbf{Adv}(break_4)$.

G5: In this game, we replace the computation of the root, chain and authentication keys $rk^{\mathsf{T}''}, ck^0_{\mathsf{T}''}, ak^{\mathsf{T}''}$ values with uniformly random and independent values $\widetilde{rk^{\mathsf{T}''}}, \widetilde{ck_0^{\mathsf{T}''}}, \widetilde{ak^{\mathsf{T}''}}$, by interacting with a sym-ss-PRFODH challenger. By this change, π_s^I and its communicating partner $\pi_{s'}^{I'}$ now have an epoch where $rk^{\mathsf{T}''}$ is uniformly random and independent from the protocol execution, and by the hardness of the sym-ss-PRFODH, $\mathcal{A}$ is unable to distinguish this change. Thus we have $\mathbf{Adv}(break_4) \leq \mathbf{Adv}^{\mathsf{sym\text{-}ss\text{-}PRFODH},\mathcal{B}_3}_{G,q,\text{KDF}}(\lambda) + \mathbf{Adv}(break_5)$.

G6: In this game, we iteratively replace the computation of $ak^{\mathsf{T}'}$, $rk^{\mathsf{T}'}$, $ck^0_{\mathsf{T}'}$ (for $\mathsf{T}'' < \mathsf{T}' \leq \mathsf{T}$) with uniformly random values $\widetilde{ak^{\mathsf{T}'}}, \widetilde{rk^{\mathsf{T}'}}, \widetilde{ck_0^{\mathsf{T}'}}$ from the same distribution in the session π_s^I (and its partner session $\pi_{s'}^{I'}$). By this change, π_s^I and its communicating partner $\pi_{s'}^{I'}$ now have authentication keys for "authenticating epoch" T and its prior epoch $\mathsf{T}-1$ where ak^{T} and $ak^{\mathsf{T}-1}$ are uniformly random and independent of the protocol execution, and by the hardness of the KDF assumption, $\mathcal{A}$ is unable to distinguish this change. Thus we have $\mathbf{Adv}(break_5) \leq (\mathsf{T}\text{-}\mathsf{T}') \cdot \mathbf{Adv}^{\mathsf{kdf},\mathcal{B}_4}_{\text{KDF}}(\lambda) + \mathbf{Adv}(break_6)$.

G7: In this game, we replace the computation of the fingerprints $\mathtt{fprint}^{T\text{-}1} = \mathsf{HMAC}(\widetilde{ak^{\mathsf{T}\text{-}1}}, H^{\mathsf{T}\text{-}1}\|\mathtt{f\_vers}\|role)$, $\mathtt{fprint}^T = \mathsf{HMAC}(\widetilde{ak^{\mathsf{T}}} = H^{\mathsf{T}}\|\mathtt{f\_vers}\|role)$, by initialising two MAC challengers to compute the fingerprints $\mathtt{fprint}^{\mathsf{T}\text{-}1}$, $\mathtt{fprint}^{\mathsf{T}}$ in the test session π_s^I. By this change, $\mathcal{A}$ has no advantage in generating a fingerprint for π_s^I such that $\pi_s^I.\alpha[\mathsf{T}] \leftarrow \mathsf{accept}$, and thus $\mathbf{Adv}(break_6) \leq 2 \cdot \mathbf{Adv}^{\mathsf{mac},\mathcal{B}_5}_{\mathsf{HMAC}}(\lambda) + \mathbf{Adv}(break_7)$.

We note here the test session has no partner with a matching session identifier (which is what the fingerprints are computed over), and that their authentication keys for T and T-1 are uniformly random and independent from the protocol flow. Thus, if $\mathcal{B}_5$ can produce a fingerprint

8. In our modelling of the Signal/MoDUSA Protocol, we assume that each message received by a session oracle is replied to with a message attached to a new ratchet key. This is both to simplify analysis, and capture the behaviour of Signal itself, which generates an ephemeral ratchet key upon receipt of a new message in a flow, even if it does not send a message immediately.

$\texttt{fprint}^{\texttt{T-1}'}$ (or $\texttt{fprint}^{\texttt{T}'}$) for the test session such that $\texttt{fprint}^{\texttt{T-1}'}$ (resp. $\texttt{fprint}^{\texttt{T}'}$) verify correctly, then $\mathcal{B}_5$ can be turned into a algorithm that breaks the mac security of HMAC. Thus: $\mathbf{Adv}(break_7) = 0$. □

6. Conclusion

User mediated protocols have long been used in real-world applications, but their security is not well understood, in part due to the difficulty of cryptographically modelling an unpredictable user and increased attack vectors. Our systematic method of modelling security in such protocols concretely captures not only traditional adversarial capabilities, but also those interacting with the device input/output capabilities. In our model, as in practice, users can check for authentication at any time, and as often as they choose. Any successful verification attempt automatically authenticates the parties involved against impersonation and MitM attacks for the entire duration of the conversation session. Signal fails to provide basic entity authentication; however, we prove that under a slight modification it could achieve quite strong guarantees. This research opens the door for analysis of traditionally out-of-scope protocol aspects. It has implications for protocols in the Internet of Things, such as Bluetooth and NFC, which rely on user interaction for device commissioning, decommissioning, authentication, and even key exchange.

References

[1] DHS/CBP/PIA-008 – Border Searches of Electronic Devices (May 2019), https://www.dhs.gov/publication/border-searches-electronic-devices

[2] Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: Security notions, proofs, and modularization for the signal protocol. In: EUROCRYPT (2019)

[3] Bella, G., Coles-Kemp, L.: Layered analysis of security ceremonies. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) Information Security and Privacy Research. pp. 273–286. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[4] Bellare, M., Singh, A.C., Jaeger, J., Nyayapati, M., Stepanovs, I.: Ratcheted encryption and key exchange: The security of messaging. pp. 619–650. LNCS, Springer, Heidelberg (2017). doi: 10.1007/978-3-319-63697-9_21

[5] Bicakci, K., Altuncu, E., Sahkulubey, M.S., Kiziloz, H.E., Uzunay, Y.: How safe is safety number? A user study on SIGNAL's fingerprint and safety number methods for public key verification. In: ISC 2018. pp. 85–98. LNCS, Springer, Heidelberg (2018). doi: 10.1007/978-3-319-99136-8_5

[6] Blazy, O., Bossuat, A., Bultel, X., Fouque, P.A., Onete, C., Pagnin, E.: SAID: Reshaping Signal into an Identity-Based Asynchronous Messaging Protocol with Authenticated Ratcheting. IACR Cryptology ePrint Archive (2019)

[7] Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. In: 2017 IEEE European Symposium on Security and Privacy (EuroS P). pp. 451–466 (2017)

[8] Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. Cryptology ePrint Archive, Report 2016/1013 (2016), http://eprint.iacr.org/2016/1013

[9] Cohn-Gordon, K., Cremers, C.J.F., Garratt, L.: On Post-compromise Security. In: IEEE 29th Computer Security Foundations Symposium, CSF 2016. pp. 164–178 (2016). doi: 10.1109/CSF.2016.19

[10] Ellison, C.: Ceremony design and analysis. Cryptology ePrint Archive, Report 2007/399 (2007), http://eprint.iacr.org/2007/399

[11] Facebook: Messenger Secret Conversations Technical Whitepaper. Tech. rep. (July 2016), https://about.fb.com/wp-content/uploads/2016/07/messenger-secret-conversations-technical-whitepaper.pdf

[12] Fischlin, M., Günther, F.: Multi-stage key exchange and the case of Google's QUIC protocol. In: Ahn, G.J., Yung, M., Li, N. (eds.) ACM CCS 14. pp. 1193–1204. ACM Press (Nov 2014). doi: 10.1145/2660267.2660308

[13] GmbH, W.S.: Key verification to secure your conversations . Tech. rep. (January 2017), https://wire.com/en/blog/key-verification-secure-conversations/

[14] GmbH, W.S.: Wire Security Whitepaper. Tech. rep. (August 2018), https://wire-docs.wire.com/download/Wire+Security+Whitepaper.pdf

[15] Hale, B.: User-mediated authentication protocols and unforgeability in key collision. In: ProvSec 2018. pp. 387–396. LNCS, Springer, Heidelberg (2018). doi: 10.1007/978-3-030-01446-9_22

[16] Høegh-Omdal, J., Kaya, C., Ottensmann, M.: The StrandHogg vulnerability (2019), https://promon.co/security-news/strandhogg/

[17] Kiayias, A., Zacharias, T., Zhang, B.: Ceremonies for end-to-end verifiable elections. In: IACR International Workshop on Public Key Cryptography. pp. 305–334. Springer (2017)

[18] Marlinspike, M., Perrin, T.: The Double Ratchet Algorithm. Tech. rep. (November 2016), https://signal.org/docs/specifications/doubleratchet/

[19] Marlinspike, M., Perrin, T.: The Signal Protocol. Tech. rep. (November 2016), https://signal.org/docs/specifications/x3dh/

[20] Marlinspike, M., Perrin, T.: The Signal Protocol: Key Compromise. Tech. rep. (November 2016), https://signal.org/docs/specifications/x3dh/#key-compromise

[21] Marlinspike, M., Perrin, T.: The X3DH Key Agreement Protocol. Tech. rep. (November 2016), https://signal.org/docs/specifications/x3dh/

[22] Martina, J.E., de Souza, T.C.S., Custodio, R.F.: Ceremonies formal analysis in pki's context. In: 2009 International Conference on Computational Science and Engineering. vol. 3, pp. 392–398. IEEE (2009)

[23] Maruyama, S., Wakabayashi, S., Mori, T.: Tap 'n ghost: A compilation of novel attack techniques against smartphone touchscreens. In: 2019 2019 IEEE Symposium on Security and Privacy (SP). pp. 628–645. IEEE Computer Society, Los Alamitos, CA, USA (may 2019). doi: 10.1109/SP.2019.00037, https://doi.ieeecomputersociety.org/10.1109/SP.2019.00037

[24] Naor, M., Rotem, L., Segev, G.: The security of lazy users in out-of-band authentication. In: Beimel, A., Dziembowski, S. (eds.) Theory of Cryptography. pp. 575–599. Springer International Publishing (2018)

[25] OpenWhisperSystems: Signal Protocol library for JavaScript. Tech. rep. (June 2019), https://github.com/signalapp/libsignal-protocol-javascript

[26] Osborne, H., Cutler, S.: Chinese border guards put secret surveillance app on tourists' phones (July 2019), https://www.theguardian.com/world/2019/jul/02/chinese-border-guards-surveillance-app-tourists-phones

[27] Poettering, B., Rösler, P.: Towards bidirectional ratcheted key exchange. pp. 3–32. LNCS, Springer, Heidelberg (2018). doi: 10.1007/978-3-319-96884-1_1

[28] Rösler, P., Mainka, C., Schwenk, J.: 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). pp. 415–429 (2018)

[29] Schröder, S., Huber, M., Wind, D., Rottermanner, C.: When SIGNAL hits the Fan: On the Usability and Security of State-of-the-Art Secure Mobile Messaging. In: Proceedings of the 1st European Workshop on Usable Security, NDSS (2016)

[30] Signal: Safety number updates. Online (November 2016), https://signal.org/blog/safety-number-updates/

[31] Signal: Signal partners with Microsoft to bring end-to-end encryption to Skype. Tech. rep. (January 2018), https://signal.org/blog/skype-partnership/

[32] Tan, J., Bauer, L., Bonneau, J., Cranor, L.F., Thomas, J., Ur, B.: Can unicorns help users compare crypto key fingerprints? In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. pp. 3787–3798. CHI '17, ACM, New York, NY, USA (2017). doi: 10.1145/3025453.3025733, http://doi.acm.org/10.1145/3025453.3025733

[33] Thomas, E.: Sydney airport seizure of phone and laptop 'alarming', say privacy groups (August 2018), https://www.theguardian.com/world/2018/aug/25/sydney-airport-seizure-of-phone-and-laptop-alarming-say-privacy-groups

[34] Vaziripour, E., Wu, J., O'Neill, M., Clinton, R., Whitehead, J., Heidbrink, S., Seamons, K., Zappala, D.: Is That You, Alice? A Usability Study of the Authentication Ceremony of Secure Messaging Applications. In: Proceedings of Conference on Usable Privacy and Security (USENIX). p. 29–47. SOUPS '17 (2017)

[35] WhatsApp: WhatsApp Encryption Overview Technical white paper. Tech. rep. (December 2017), https://www.cdn.whatsapp.net/security/WhatsApp-Security-Whitepaper.pdf

Appendix A. Details of the Signal Protocol

In this section, we describe the initial key exchange and the asymmetric ratchet of the Signal protocol in detail, complete with protocol flow and key schedule diagrams. We will be modifying the Signal key schedule to generate new *authentication keys* that will be used in our Modified Signal Authentication Protocol MoDUSA. We begin by describing the so-called X3DH key exchange, used to generate the initial secret values for Alice and Bob to begin communication.

Recall that in Signal, when a device A owned by Alice wishes to communicate with another device B owned by Bob, in practice A fetches what is known as a prekey bundle, containing a set of public keys that were (in theory) generated by B, and uses these values to generate the initial message sent from A to A. Each prekey bundle contains the following:

- `regId`: a 32-bit integer value, generated by each device I during registration.
- `deviceId`: a 32-bit integer value that distinguishes between prekey bundles for each device that the user maintains.[9]
- `preKeyPublic`: a one-time-use DH public key, generated frequently by the device I, which we denote with $otpk_I$. [10]
- `signedPreKeyPublic`: the medium-term DH public key, generated at regular intervals, which we denote $sppk_I$.
- `signedPreKeySignature`: the signature σ_I over $sppk_I$ using the identity key idk_I.
- `identityKey`: the long-term DH public key of the responder, generated during registration, which we denote $idpk_I$.
- `protocolAddress`: a `regID` and `deviceID` pair.

9. In our representation of the Signal X3DH key exchange in Figure 7, we pair the `regId` with the `deviceId` together as ID_I for conciseness.

10. The prekeys (both signed and unsigned) are paired with a prekey identifier, which we have omitted here for conciseness.

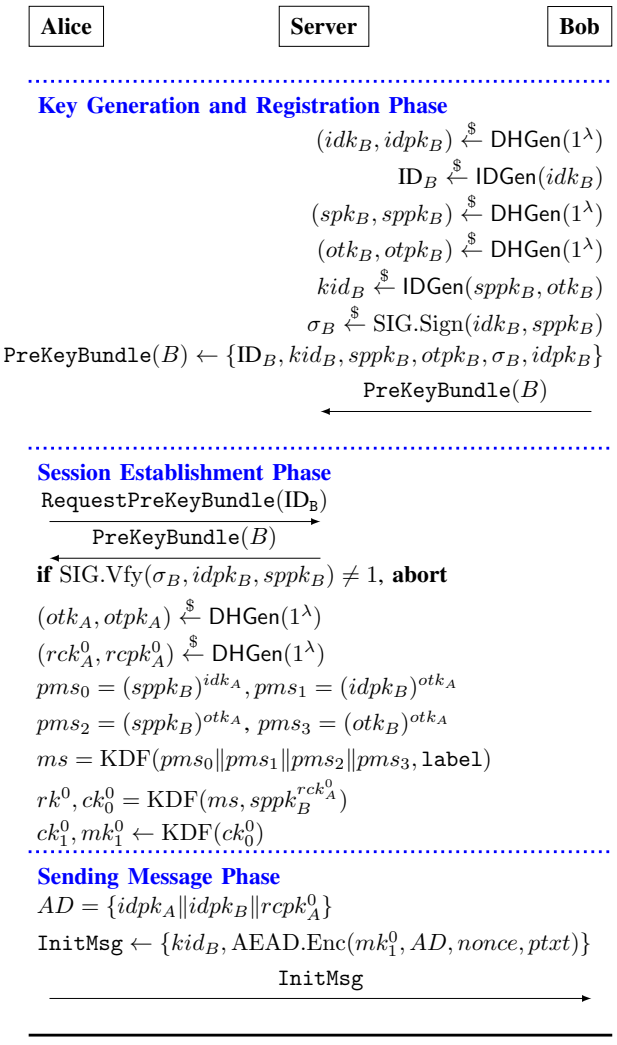

Figure 7. A protocol flow describing the Signal X3DH initial key exchange protocol. In this protocol execution, user Bob has generated a `PreKeyBundle` locally, and stored the public key values (and identifiers) with the Server. At some point, user Alice will request the `PreKeyBundle` and use it to establish a message key mk, encrypting a message and sending the ciphertext to Bob. IDGen is a function that takes either the identity public key idk or the public prekeys $sppk, otpk$ and generates the tuple (`regId`, `deviceId`) (or the key identifiers for the prekeys, respectively).

Alice then generates a one-time keypair $(otk_A, otpk_A)$, and with the long-term identity keypair $(idk_A, idpk_A)$, derives the following values:

$$pms_0 = (sppk_B)^{idk_A}, \ pms_1 = (idpk_B)^{otk_A}$$
$$pms_2 = (sppk_B)^{otk_A}, \ pms_3 = (otk_B)^{otk_A}$$

Alice can now compute the master secret ms, where $ms = \text{KDF}(pms_0 \| pms_1 \| pms_2 \| pms_3)$, and generates the first ratchet keypair $(rck_0^A, rcpk_0^A)$ to derive the first root and chain key $(rk^0, ck_0^0) \leftarrow \text{KDF}(ms, (sppk_B)^{rck_0^A})$. Finally, Alice computes the next chain key and the first message key $ck_1^0, mk_1^0 \leftarrow \text{KDF}(ck_0^0)$.

Alice uses an AEAD symmetric cipher to encrypt a plaintext message $ptxt$. The additional data field AD is set as the concatenation of the identity public keys of both Alice and Bob, as well as Alice's first ratchet key, i.e. $AD = \{idpk_A \| idpk_B \| rcpk_A^0\}$. Finally, Alice sends the ciphertext $c_1^0 = \text{AEAD.Encrypt}(mk_1^0, AD, nonce, ptxt)$

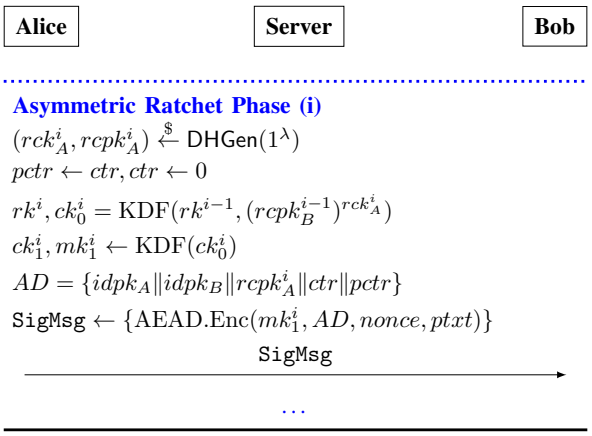

Figure 8. A protocol flow describing the Signal Double Ratchet protocol. In this protocol execution, user Alice has received the $(i-1)$-th ratchet public key $rcpk_B^{i-1}$, and wants to send a new message. Afterwards, Alice has not yet received a new ratchet key from Bob, and thus will engage in a symmetric ratchet, generate a new chain and message key and message key, encrypt a message and send the ciphertext to Bob. Note that ctr and $pctr$ are both counters - ctr maintains the number of messages the user has sent in this chain of messages, while $pctr$ maintains the number of messages the user sent in the previous chain of messages.

and sends $\{kid_B, c_1^0\}$ to Bob, where kid_B are some identifiers for the signed prekey and the one-time key.

Appendix B. MoDUSA Usability

When comparing the MoDUSA protocol to the original Signal authentication protocol, it is clear that a potential impediment to usability would come from users being required to pick between two fingerprints when comparing numeric codes. This requires users to check the index associated with both fingerprints (on both devices) and compare the *highest*-indexed fingerprint that both devices have. This can be streamlined (for instance, a two-step mechanism where users first select a button that displays the highest index that both devices have, and then the device afterwards will displays only a single fingerprint), but any type of barrier will cause certain types of end-users to disregard this authentication mechanism. Some interesting future work would be to improve our suggested solution to avoid presenting this additional complexity to users of numeric comparison.

It is interesting to note that for QR code-based fingerprints, this dual-fingerprint problem does not occur. Instead, the QR code can contain *both* fingerprints, and the scanning device need only check that one of the fingerprints match, simplifying the authentication mechanism significantly for users.

In addition, to make collision attacks on fingerprints more difficult, our solution does not truncate the numeric code fingerprints. In practice, it is unlikely that users will actually compare the full fingerprints, depending on how they are encoded or represented. A possible direction to follow is visual representations of fingerprints [32] that may prevent user-driven truncation.

Another impediment to usability is the fact that the MoDUSA fingerprints change depending on the epoch in which the user authenticates. In comparison, the Signal authentication protocol fingerprints (since they only contain identifier and public-key information) are stable. Stable fingerprints allow for Alice to host her fingerprint on a publicly-accessible resource, and she only needs to upload it once, never updating it. Of course, this leads to the breaks in security previously mentioned. However, from a usability perspective, a MoDUSA fingerprint is only useful for authenticating ratchets up to the specific epoch – if Bob accesses an old fingerprint without having saved the authentication key for that epoch, Bob cannot recompute the fingerprint locally and use it to detect an active adversary. Thus any OOB channel using MoDUSA fingerprints must be able to update as frequently as comparison is expected (note that this need not be per epoch, but does require storage of authentication keys between expected comparisons). It is important to note that in MoDUSA fingerprints (whether QR code or numeric codes) are only required to be computed when authenticating – it is not necessary to constantly "evolve" the fingerprints with each epoch.

Computational and Storage Costs In the original Signal authentication protocol, since the fingerprints are generated via an iterative hash, each fingerprint takes 10,400 hash operations to generate (5,200 hash computations for each user's half of the fingerprint). However, this is only required once: from then on, users need only store the 60 digit fingerprint / QR code and retrieve the fingerprints whenever they use the Signal authentication protocol. This differs from MoDUSA, where users are required to either store a transcript (containing each of the public keys sent between devices) or two rolling hash values (if the implementation uses updating hash values), as well as two authentication keys (for the current and previous epoch) and a counter containing the current index of the epoch. Computationally, MoDUSA requires two hash operations and two MAC operations whenever generating a fingerprint (again, this being on demand and not necessarily per epoch), but the upfront cost is lower – there is no need to perform 5,200 hash operations to generate the initial fingerprint.

Appendix C. Flaws in the Signal Authentication Protocol

Here we discuss further weaknesses in the Signal Authentication Protocol that mean that we cannot prove meaningful guarantees within the *META* security model.

C.1. Identity key collision attacks

Recalling the details of Figure 1, it is clear that the "displayable fingerprint", or numeric codes read by the users is significantly truncated from the original hash output that contains both users' identifiers and identity keys. Indeed, the truncation process immediately discards the final 272 bits of both the `local_fprint` and the `remote_fprint`. Thus, if one can compute another fingerprint $\mathtt{fprint} = H_i(0 \| \mathtt{fvers} \| idpk_E \| \mathrm{ID}_E, idpk_E)$ such that $[\mathtt{remote\_fprint}]_0^{239} = [\mathtt{fprint}]_0^{239}$ (i.e. cause a collision between the first 240 bits of the remote user's fingerprint), then a malicious E can inject a maliciously controlled `PreKeyBundle` to one of the parties and still

get the two partners to accept the verification of their fingerprints without detection, even if the users are acting completely honestly with respect to the expected user behaviours.

Due to the further truncation described in Figure 1 (specifically, reducing the 40-bit "blocks" to integers modulo 100,000), it is even easier for the attacker: E only requires a collision on the specific 102 bits that are represented in the remote fingerprint itself[11]. By the birthday bound and iteratively generating new identity keys and computing fingerprints using the same identifiers, an attacker can produce a pair of colliding fingerprints for distinct identity keys with 50% probability by generating only 2^{51} long-term keys. Note that this method can then be used to attack *any* conversation that uses these colliding identity keys. This does not allow the attacker to compute colliding identity keys with specific targets, but we argue that the ability of an attacker to compute any colliding keys constitutes a valid attack against the Signal authentication protocol.

11. Precisely, the last 17 bits of each of the first six 40-bit "chunks" of the hashed output of the fingerprint.