Optimal Channel Security Against Fine-Grained State Compromise: The Safety of Messaging

Joseph Jaeger and Igors Stepanovs

Department of Computer Science and Engineering, University of California San Diego, La Jolla, USA. `{jsjaeger,istepano}@eng.ucsd.edu`

Abstract. We aim to understand the best possible security of a (bidirectional) cryptographic channel against an adversary that may arbitrarily and repeatedly learn the secret state of either communicating party. We give a formal security definition and a proven-secure construction. This construction provides better security against state compromise than the Signal Double Ratchet Algorithm or any other known channel construction. To facilitate this we define and construct new forms of public-key encryption and digital signatures that update their keys over time.

1 Introduction

End-to-end encrypted communication is becoming a usable reality for the masses in the form of secure messaging apps. However, chat sessions can be extremely long-lived and their secrets are stored on end user devices, so they are particularly vulnerable to having their cryptographic secrets exfiltrated to an attacker by malware or physical access to the device. The Signal protocol [33] by Open Whisper Systems tries to mitigate this threat by continually updating the key used for encryption. Beyond its use in the Signal messaging app, this protocol has been adopted by a number of other secure messaging apps. This includes being used by default in WhatsApp and as part of secure messaging modes of Facebook Messenger, Google Allo, and Skype.

WhatsApp alone has 1 billion daily active users [43]. It is commonly agreed in the cryptography and security community that the Signal protocol is secure. However, the protocol was designed without an explicitly defined security notion. This raises the questions: what security does it achieve and could we do better?

In this work we study the latter question, aiming to understand the best possible security of two-party communication in the face of state exfiltration. We formally define this notion of security and design a scheme that provably achieves it.

Security against compromise. When a party's secret state is exposed we would like both that the security of past messages and (as soon as possible) the security of future messages not be damaged. These notions have been considered in a variety of contexts with differing terminology. The systemization of knowledge paper on secure messaging [42] by Unger, Dechand, Bonneau, Fahl, Perl, Goldberg, and Smith evaluates and systematizes a number of secure messaging systems. In it they describe a variety of terms for these types of security including "forward secrecy," "backwards secrecy," "self-healing," and "future secrecy" and note that they are "controversial and vague." Cohn-Gordon, Cremers, and Garratt [15] study the future direction under the term of post-compromise security and similarly discuss the terms "future secrecy," "healing," and "bootstrapping" and note that they are "intuitive" but "not well-defined." Our security notion intuitively captures any of these informal terms, but we avoid using any of them directly by aiming generically for the best possible security against compromise.

Channels. The standard model for studying secure two party communication is that of the (cryptographic) channel. The first attempts to consider the secure channel as a cryptographic object were made by Shoup [39] and Canetti [11]. It was then formalized by Canetti and Krawczyk [13] as a modular way to combine a key exchange protocol with authenticated encryption, which covers both privacy and integrity. Krawczyk [28] and Namprempre [32] study what are the necessary and sufficient security notions to build a secure channel from these primitives.

Modern definitions of channels often draw from the game-based notion of security for stateful authenticated-encryption as defined by Bellare, Kohno, and Namprempre [4]. We follow this convention which assumes initial generation of keys is trusted. In addition to requiring that a channel provides integrity and privacy of the encrypted data, we will require integrity for associated data as introduced by Rogaway [36].

Recently Marson and Poettering [30] closed a gap in the modeling of two-party communication by capturing the bidirectional nature of practical channels in their definitions. We work with their notion of bidirectional channels because it closely models the behavior desired in practice and the bidirectional nature of communication allows us to achieve a fine-grained security against compromise.

Definitional contributions. This paper aims to specify and achieve the best possible security of a bidirectional channel against state compromise. We provide a formal, game-based definition of security and a construction that provably achieves it. We analyze our construction in a concrete security framework [2] and give precise bounds on the advantage of an attacker.

To derive the best possible notion of security against state compromise we first specify a basic input-output interface via a game that describes how the adversary interacts with the channel. This corresponds roughly to combining the integrity and confidentiality games of [30] and adding an oracle that returns the secret state of a specified user to the adversary. Then we specify several attacks that break the security of *any* channel. We define our final security notion by minimally extending the initial interface game to disallow these unavoidable attacks while allowing all other behaviors. Our security definition is consequently the best possible with respect to the specified interface because our attacks rule out the possibility of any stronger notion.

One security notion is an all-in-one notion in the style of [37] that simultaneously requires integrity and privacy of the channel. It asks for the maximal possible security in the face of the exposure of either party's state. A surprising requirement of our definition is that given the state of a user the adversary should not be able to decrypt ciphertexts sent by that user or send forged ciphertexts to that user.

Protocols that update their keys. The OTR (Off-the-Record) messaging protocol [10] is an important predecessor to Signal. It has parties repeatedly exchange Diffie-Hellman elements to derive new keys. The Double Ratchet Algorithm of Signal uses a similar Diffie-Hellman update mechanism and extends it by using a symmetric key-derivation function to update keys when there is no Diffie-Hellman update available. Both methods of updating keys are often referred to as ratcheting (a term introduced by Langley [29]). While the Double Ratchet Algorithm was explicitly designed to achieve strong notions of security against state compromise with respect to privacy, the designers explicitly consider security against a passive eavesdropper [21]; authenticity in the face of compromise is out of scope.

The first academic security analysis of Signal was due to Cohn-Gordan, Cremers, Dowling, Garratt, and Stebila [14]. They only considered the security of the key exchange underlying the Double Ratchet Algorithm and used a security definition explicitly tailored to understanding its security instead of being widely applicable to any scheme.

Work by Bellare, Camper Singh, Jaeger, Nyayapati, and Stepanovs [7] sought to formally understand ratcheting as an independent primitive, introducing the notions of (one-directional) ratcheted key exchange and ratcheted encryption. In their model a compromise of the receiving party's secrets permanently and irrevocably disrupts all security, past and future. Further they strictly separate the exchange of key update information from the exchange of messages. Such a model cannot capture a protocol like the Double Ratchet Algorithm for which the two are inextricably combined. On the positive side, they did explicitly model authenticity in the face of compromise.

In [26], Günther and Mazaheri study a key update mechanism introduced in TLS 1.3. Their security definition treats update messages as being out-of-band and thus implicitly authenticated. Their definition is clearly tailored to understand TLS 1.3 specifically.

Instead of analyzing an existing scheme, we strive to understand the best possible security with respect to both privacy and authenticity in the face of state compromise. The techniques we use to achieve this differ from those underlying the schemes discussed above, because all of them rely on exchanging information to create a shared symmetric key that is ultimately used for encryption. Our security notion is not achievable by a scheme of this form and instead requires that asymmetric primitives be used throughout.

Consequently, our scheme is more computationally intensive than those mentioned above. However, as a part of OTR or the Double Ratchet Algorithm, when users are actively sending messages back and forth (the case where efficiency is most relevant), they will be performing asymmetric Diffie-Hellman based key updates prior to most message encryptions. This indicates that the overhead of extra computation with asymmetric techniques is not debilitating in our motivating context of secure messaging. However, the asymmetric techniques we require are likely less efficient than Diffie-Hellman computations so we do not currently know whether our scheme meets realistic efficiency requirements.

Our construction. Our construction of a secure channel is given in Section 6.1. It shows how to generically build the channel from a collision-resistant hash function, a public-key encryption scheme, and a digital signature scheme. The latter two require new versions of the primitives that we describe momentarily.

The hash function is used to store transcripts of the communication in the form of hashes of all sent or received ciphertexts. These transcripts are included as part of every ciphertext and a user will not accept a ciphertext with transcripts that do not match those it has stored locally. Every ciphertext sent by a user is signed by their current digital signature signing key and includes the verification key corresponding to their next signing key. Similarly a user will include a new encryption key with every ciphertext they send. The sending user will use the most recent encryption key they have received from the other user and the receiving user will delete all decryption keys that are older than the one most recently used by the sender.

New notions of public-key encryption and digital signatures. Our construction uses new forms of public-key encryption and digital signatures that update their keys over time, which we define in Section 3. The former updates its keys with every ciphertext. We refer to it as key-updating public-key encryption. The latter includes extra algorithms that allow the keys to be updated with respect to an arbitrary string. We refer to it as key-updatable digital signature schemes. In our construction a user updates their signing key with their transcript every time they receive a ciphertext.

For public-key encryption we consider encryption with labels and require an IND-CCA style security be maintained even if the adversary is given the decryption key after all challenge ciphertexts have been decrypted or an adversarially generated ciphertext has been decrypted. We show how to construct such scheme from hierarchical identity-based encryption [23].

For digital signatures, security requires that an adversary is unable to forge a signature even given the signing key as long as the sequence of strings used to update it is not a prefix of the sequence of strings used to update the verification key. We additionally require that the scheme has unique signatures (i.e. for any sequence of updates and any message an adversary can only find one signature that will verify). We show how to construct this from a digital signature scheme that is forward secure [5] and has unique signatures.

Related work. Several works [22, 9] extended the definitions of channels to address the stream-based interface provided by channels like TLS, SSH, and QUIC. Our primary motivation is to build a channel for messaging where an atomic interface for messages is more appropriate.

Numerous areas of research within cryptography are motivated by the threat of key compromise. These include key-insulated cryptography [18–20], secret sharing [38, 31, 41], threshold cryptography [16], proactive cryptography [34], and forward security [25, 17]. Forward security, in particular, was introduced in the context of key-exchange [25, 17] but has since been considered for a variety of primitives including symmetric [8] and asymmetric encryption [12] and digital signature schemes [5]. Green and Miers [24] propose using puncturable encryption for forward secure asynchronous messaging.

In concurrent and independent work, Poettering and Rösler [35] extend the definitions of ratcheted key exchange from [7] to be bidirectional. Their security definition is conceptually similar to our definition for bidirectional channels because both works aim to achieve strong notions of security against an adversary that can arbitrarily and repeatedly learn the secret state of either communicating party. In constructing a secure ratcheted key exchange scheme they make use of a key-updatable key encapsulation mechanism (KEM), a new primitive they introduce in their work. The key-updatable nature of this is conceptually similar to that of the key-updatable digital signature schemes we introduce in our work. To construct such a KEM they make use of hierarchical identity-based encryption in a manner similar to how we construct key-updating public-key encryption. The goal of their work differs from ours; they only consider security for the exchange of symmetric keys while we do so for the exchange of messages.

2 Preliminaries

Notation and conventions. Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of non-negative integers. Let ε denote the empty string. If $x \in \{0,1\}^*$ is a string then $|x|$ denotes its length. By $x \,\|\, y$ we denote the concatenation of strings x and y. If X is a finite set, we let $x \leftarrow_\$ X$ denote picking an element of X uniformly at random and assigning it to x. By $(X)^n$ we denote the n-ary Cartesian product of X. We let $x_1 \leftarrow x_2 \leftarrow \cdots \leftarrow x_n \leftarrow v$ denote assigning the value v to each variable x_i for $i = 1, \ldots, n$.

If **mem** is a table, we use $\mathbf{mem}[p]$ to denote the element of the table that is indexed by p. By $\mathbf{mem}[0, \ldots, \infty] \leftarrow v$ we denote initializing all elements of **mem** to v. For $a, b \in \mathbb{N}$ we let $v \leftarrow \mathbf{mem}[a, \ldots, b]$ denote setting v equal to the tuple obtained by removing all $\bot$ elements from $(\mathbf{mem}[a], \mathbf{mem}[a+1], \ldots, \mathbf{mem}[b])$. It is the empty vector $()$ if all of these table entries are $\bot$ or if $a > b$. A tuple

```
Game $\mathrm{CR}_{\mathsf{H}}^{\mathcal{A}_{\mathsf{H}}}$
$hk \leftarrow\!\!\$\ \mathsf{H.Kg}$
$(m_0, m_1) \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{H}}(hk)$
$y_0 \leftarrow \mathsf{H.Ev}(hk, m_0)$
$y_1 \leftarrow \mathsf{H.Ev}(hk, m_1)$
Return $(m_0 \neq m_1)$ and $(y_0 = y_1)$
```

```
Game $\mathrm{UNIQ}_{\mathsf{DS}}^{\mathcal{B}_{\mathsf{DS}}}$
$(\Lambda, m, \sigma_1, \sigma_2, \boldsymbol{\Delta}) \leftarrow\!\!\$\ \mathcal{B}_{\mathsf{DS}}^{\textsc{NewUser}}$
$(sk[\Lambda], vk[\Lambda]) \leftarrow \mathsf{DS.Kg}(z[\Lambda])$
$(vk, t_1) \leftarrow \mathsf{DS.Vrfy}(vk[\Lambda], \sigma_1, m, \boldsymbol{\Delta})$
$(vk, t_2) \leftarrow \mathsf{DS.Vrfy}(vk[\Lambda], \sigma_2, m, \boldsymbol{\Delta})$
Return $t_1$ and $t_2$ and $\sigma_1 \neq \sigma_2$

$\textsc{NewUser}(\Lambda)$   // $\Lambda \in \{0,1\}^*$
If $z[\Lambda] \neq \bot$ then return $\bot$
$z[\Lambda] \leftarrow\!\!\$\ \mathsf{DS.KgRS}$ ; Return $z[\Lambda]$
```

Fig. 1. Games defining collision-resistance of function family H and signature uniqueness of key-updatable digital signature scheme DS.

$\boldsymbol{x} = (x_1, \dots)$ specifies a uniquely decodable concatenation of strings $x_1, \dots$. We say $\boldsymbol{x} \sqsubseteq \boldsymbol{y}$ if $\boldsymbol{x}$ is a prefix of $\boldsymbol{y}$. More formally, $(x_1, \dots, x_n) \sqsubseteq (y_1, \dots, y_m)$ if $n \leq m$ and $x_i = y_i$ for all $i \in \{1, \dots, n\}$.

Algorithms may be randomized unless otherwise indicated. Running time is worst case. If A is an algorithm, we let $y \leftarrow A(x_1, \dots; r)$ denote running A with random coins r on inputs $x_1, \dots$ and assigning the output to y. Any state maintained by an algorithm will explicitly be shown as input and output of that algorithm. We let $y \leftarrow\!\!\$\ A(x_1, \dots)$ denote picking r at random and letting $y \leftarrow A(x_1, \dots; r)$. We omit the semicolon when there are no inputs other than the random coins. We let $[A(x_1, \dots)]$ denote the set of all possible outputs of A when invoked with inputs $x_1, \dots$. Adversaries are algorithms. The instruction **abort**$(x_1, \dots)$ is used by an adversary to immediately halt with output $(x_1, \dots)$.

We use a special symbol $\bot \notin \{0,1\}^*$ to denote an empty table position, and we also return it as an error code indicating an invalid input. An algorithm may not accept $\bot$ as input. If $x_i = \bot$ for some i when executing $(y_1, \dots) \leftarrow A(x_1 \dots)$ we assume that $y_j = \bot$ for all j. We assume that adversaries never pass $\bot$ as input to their oracles.

We use the code based game playing framework of [6]. (See Fig. 1 for an example of a game.) We let $\Pr[\mathrm{G}]$ denote the probability that game G returns true. In code, tables are initially empty. We adopt the convention that the running time of an adversary means the worst case execution time of the adversary in the game that executes it, so that time for game setup steps and time to compute answers to oracle queries is included.

Function families. A family of functions H specifies algorithms $\mathsf{H.Kg}$ and $\mathsf{H.Ev}$, where $\mathsf{H.Ev}$ is deterministic. Key generation algorithm $\mathsf{H.Kg}$ returns a key hk. Evaluation algorithm $\mathsf{H.Ev}$ takes hk and an input $x \in \{0,1\}^*$ to return an output y, denoted by $y \leftarrow \mathsf{H.Ev}(hk, x)$.

Collision-resistant functions. Consider game CR of Fig. 1 associated to a function family H and an adversary $\mathcal{A}_{\mathsf{H}}$. The game samples a random key hk for function family H. In order to win the game, adversary $\mathcal{A}_{\mathsf{H}}$ has to find two distinct messages m_0, m_1 such that $\mathsf{H.Ev}(hk, m_0) = \mathsf{H.Ev}(hk, m_1)$. The advantage of $\mathcal{A}_{\mathsf{H}}$ in breaking the CR security of H is defined as $\mathsf{Adv}^{\mathsf{cr}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{H}}) = \Pr[\mathrm{CR}^{\mathcal{A}_{\mathsf{H}}}_{\mathsf{H}}]$.

Digital signature schemes. A digital signature scheme DS specifies algorithms $\mathsf{DS.Kg}$, $\mathsf{DS.Sign}$ and $\mathsf{DS.Vrfy}$, where $\mathsf{DS.Vrfy}$ is deterministic. Associated to DS is a key generation randomness space $\mathsf{DS.KgRS}$ and signing algorithm's randomness space $\mathsf{DS.SignRS}$. Key generation algorithm $\mathsf{DS.Kg}$ takes randomness $z \in \mathsf{DS.KgRS}$ to return a signing key sk and a verification key vk, denoted by $(sk, vk) \leftarrow \mathsf{DS.Kg}(z)$. Signing algorithm $\mathsf{DS.Sign}$ takes sk, a message $m \in \{0,1\}^*$ and randomness $z \in \mathsf{DS.SignRS}$ to return a signature σ, denoted by $\sigma \leftarrow \mathsf{DS.Sign}(sk, m; z)$. Verification algorithm $\mathsf{DS.Vrfy}$ takes vk, σ, and m to return a decision $t \in \{\mathsf{true}, \mathsf{false}\}$ regarding whether σ is a valid signature of m under vk, denoted by $t \leftarrow \mathsf{DS.Vrfy}(vk, \sigma, m)$. The correctness condition for DS requires that $\mathsf{DS.Vrfy}(vk, \sigma, m) = \mathsf{true}$ for all $(sk, vk) \in [\mathsf{DS.Kg}]$, all $m \in \{0,1\}^*$, and all $\sigma \in [\mathsf{DS.Sign}(sk, m)]$.

We define the min-entropy of algorithm $\mathsf{DS.Kg}$ as $\mathrm{H}_\infty(\mathsf{DS.Kg})$, such that

$$2^{-\mathrm{H}_\infty(\mathsf{DS.Kg})} = \max_{vk} \Pr\left[vk^* = vk : (sk^*, vk^*) \leftarrow_{\$} \mathsf{DS.Kg}\right].$$

The probability is defined over the random coins used for $\mathsf{DS.Kg}$. Note that the min-entropy is defined with respect to verification keys, regardless of the corresponding values of the secret keys.

3 New asymmetric primitives

In this section we define key-updatable digital signatures and key-updating public-key encryption. The former allows its keys to be updated with arbitrary strings. The latter updates its keys with every ciphertext that is sent/received. While in general one would prefer the size of keys, signatures, and ciphertexts to be constant we will be willing to accept schemes for which these grow linearly in the number of updates. As we will discuss later, these are plausibly acceptable inefficiencies for our use cases.

We specify multi-user security definitions for both primitives, because it allows tighter reductions when we construct a channel from these primitives. Single-user variants of these definitions are obtained by only allowing the adversary to interact with one user and can be shown to imply the multi-user versions by a standard hybrid argument. Starting with [1] constructions have been given for a variety of primitives that allow multi-user security to be proven without the factor q security loss introduced by a hybrid argument. If analogous constructions can be found for our primitives then our results will give tight bounds on the security of our channel.

```
Game $\mathrm{DSCORR}^{\mathcal{C}}_{\mathsf{DS}}$
$i \leftarrow 0$ ; $\nu \leftarrow\!\!\$\ \mathsf{DS.KgRS}$
$(sk, vk) \leftarrow \mathsf{DS.Kg}(\nu)$
$\mathcal{C}^{\textsc{Upd},\textsc{Sign}}(\nu)$
Return $\mathsf{bad}$

$\underline{\textsc{Upd}(\varDelta)}$  // $\varDelta \in \{0,1\}^*$
$i \leftarrow i+1$ ; $\varDelta_i \leftarrow \varDelta$
$sk \leftarrow\!\!\$\ \mathsf{DS.UpdSk}(sk, \varDelta)$
Return $sk$

$\underline{\textsc{Sign}(m)}$  // $m \in \{0,1\}^*$
$\sigma \leftarrow\!\!\$\ \mathsf{DS.Sign}(sk, m)$
$\boldsymbol{\varDelta} \leftarrow (\varDelta_1, \ldots, \varDelta_i)$
$(vk^*, t) \leftarrow \mathsf{DS.Vrfy}(vk, \sigma, m, \boldsymbol{\varDelta})$
If not $t$ then $\mathsf{bad} \leftarrow \mathsf{true}$
```

```
Game $\mathrm{PKECORR}^{\mathcal{C}}_{\mathsf{PKE}}$
$\nu \leftarrow\!\!\$\ \mathsf{PKE.KgRS}$
$(ek, dk) \leftarrow \mathsf{PKE.Kg}(\nu)$
$\mathcal{C}^{\textsc{Enc}}(\nu)$
Return $\mathsf{bad}$

$\underline{\textsc{Enc}(m, \ell)}$  // $m, \ell \in \{0,1\}^*$
$(ek, c) \leftarrow\!\!\$\ \mathsf{PKE.Enc}(ek, \ell, m)$
$(dk, m') \leftarrow\!\!\$\ \mathsf{PKE.Dec}(dk, \ell, c)$
If $m' \neq m$ then $\mathsf{bad} \leftarrow \mathsf{true}$
Return $(ek, dk)$
```

Fig. 2. Games defining correctness of key-updatable digital signature scheme DS and correctness of key-updating public-key encryption scheme PKE.

3.1 Key-updatable digital signature schemes

We start by formally defining the syntax and correctness of a key-updatable digital signature scheme. Then we specify a security definition for it. We will briefly sketch how to construct such a scheme, but leave the details to [27].

Syntax and correctness. A *key-updatable* digital signature scheme is a digital signature scheme with additional algorithms $\mathsf{DS.UpdSk}$ and $\mathsf{DS.UpdVk}$, where $\mathsf{DS.UpdVk}$ is deterministic. Signing-key update algorithm $\mathsf{DS.UpdSk}$ takes a signing key sk and a key update information $\varDelta \in \{0,1\}^*$ to return a new signing key sk, denoted by $sk \leftarrow\!\!\$\ \mathsf{DS.UpdSk}(sk, \varDelta)$. Verification-key update algorithm $\mathsf{DS.UpdVk}$ takes a verification key vk and a key update information $\varDelta \in \{0,1\}^*$ to return a new verification key vk, denoted by $vk \leftarrow \mathsf{DS.UpdVk}(vk, \varDelta)$.

For compactness, when $\boldsymbol{\varDelta} = (\varDelta_1, \varDelta_2, \ldots)$ we sometimes write $(vk, t) \leftarrow \mathsf{DS.Vrfy}(vk, \sigma, m, \boldsymbol{\varDelta})$ to denote updating the key via $vk \leftarrow \mathsf{DS.UpdVk}(vk, \varDelta_i)$ for $i = 1, \ldots, n$ and then evaluating $t \leftarrow \mathsf{DS.Vrfy}(vk, \sigma, m)$.

The key-update correctness condition requires that signatures must verify correctly as long as the signing and the verification keys are both updated with the same sequence of key update information $\boldsymbol{\varDelta} = (\varDelta_1, \varDelta_2, \ldots)$. To formalize this, consider game DSCORR of Fig. 2, associated to a key-updatable digital signature scheme DS and an adversary $\mathcal{C}$. The advantage of an adversary $\mathcal{C}$ against the correctness of DS is given by $\mathsf{Adv}^{\mathsf{dscorr}}_{\mathsf{DS}}(\mathcal{C}) = \Pr[\mathrm{DSCORR}^{\mathcal{C}}_{\mathsf{DS}}]$. We require that $\mathsf{Adv}^{\mathsf{dscorr}}_{\mathsf{DS}}(\mathcal{C}) = 0$ for all (even unbounded) adversaries $\mathcal{C}$. See Section 4 for discussion on game-based definitions of correctness.

```
Game $\mathrm{UFEXP}_{\mathsf{DS}}^{\mathcal{A}_{\mathsf{DS}}}$
$\mathsf{out} \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{DS}}^{\textsc{NewUser},\textsc{Upd},\textsc{Sign},\textsc{Exp}}$
$(\Lambda, \sigma, m, \boldsymbol{\Delta}) \leftarrow \mathsf{out}$
$\mathsf{forgery} \leftarrow (\sigma, m, \boldsymbol{\Delta})$
$\mathsf{given} \leftarrow (\sigma^*[\Lambda], m^*[\Lambda], \boldsymbol{\Delta}^*[\Lambda])$
$t_1 \leftarrow (\mathsf{forgery} = \mathsf{given})$
$t_2 \leftarrow \boldsymbol{\Delta}'[\Lambda] \sqsubseteq \boldsymbol{\Delta}$
$\mathsf{cheated} \leftarrow (t_1 \text{ or } t_2)$
$vk \leftarrow vk[\Lambda]$
$(vk, \mathsf{win}) \leftarrow \mathsf{DS.Vrfy}(vk, \sigma, m, \boldsymbol{\Delta})$
Return $\mathsf{win}$ and not $\mathsf{cheated}$

$\textsc{NewUser}(\Lambda)$   // $\Lambda \in \{0,1\}^*$
If $vk[\Lambda] \neq \bot$ then return $\bot$
$i[\Lambda] \leftarrow 0$
$\boldsymbol{\Delta}^*[\Lambda] \leftarrow \bot$ ; $\boldsymbol{\Delta}'[\Lambda] \leftarrow \bot$
$(sk[\Lambda], vk[\Lambda]) \leftarrow\!\!\$\ \mathsf{DS.Kg}$
Return $vk[\Lambda]$

$\textsc{Upd}(\Lambda, \Delta)$   // $\Lambda, \Delta \in \{0,1\}^*$
If $sk[\Lambda] = \bot$ then return $\bot$
$i[\Lambda] \leftarrow i[\Lambda] + 1$ ; $\Delta_{i[\Lambda]}[\Lambda] \leftarrow \Delta$
$sk[\Lambda] \leftarrow\!\!\$\ \mathsf{DS.UpdSk}(sk[\Lambda], \Delta)$
Return $\bot$

$\textsc{Sign}(\Lambda, m)$   // $\Lambda, m \in \{0,1\}^*$
If $sk[\Lambda] = \bot$ then return $\bot$
$\sigma \leftarrow\!\!\$\ \mathsf{DS.Sign}(sk[\Lambda], m)$
$sk[\Lambda] \leftarrow \bot$
$(\sigma^*[\Lambda], m^*[\Lambda]) \leftarrow (\sigma, m)$
$\boldsymbol{\Delta}^*[\Lambda] \leftarrow (\Delta_1[\Lambda], \ldots, \Delta_{i[\Lambda]}[\Lambda])$
Return $\sigma$

$\textsc{Exp}(\Lambda)$   // $\Lambda \in \{0,1\}^*$
If $sk[\Lambda] = \bot$ then return $\bot$
If $\boldsymbol{\Delta}'[\Lambda] = \bot$ then
    $\boldsymbol{\Delta}'[\Lambda] \leftarrow (\Delta_1[\Lambda], \ldots, \Delta_{i[\Lambda]}[\Lambda])$
Return $sk[\Lambda]$
```

```
Game $\mathrm{INDEXP}_{\mathsf{PKE}}^{\mathcal{A}_{\mathsf{PKE}}}$
$b \leftarrow\!\!\$\ \{0,1\}$
$b' \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{PKE}}^{\textsc{NewUser},\textsc{Enc},\textsc{Dec},\textsc{ExpRand},\textsc{ExpDk}}$
Return $b = b'$

$\textsc{NewUser}(\Lambda)$   // $\Lambda \in \{0,1\}^*$
If $dk[\Lambda] \neq \bot$ or $\mathsf{nextop} \neq \bot$ then return $\bot$
$s[\Lambda] \leftarrow 0$ ; $r[\Lambda] \leftarrow 0$ ; $\mathsf{exp}[\Lambda] \leftarrow \mathsf{false}$
$z[\Lambda] \leftarrow\!\!\$\ \mathsf{PKE.EncRS}$ ; $(ek[\Lambda], dk[\Lambda]) \leftarrow\!\!\$\ \mathsf{PKE.Kg}$
Return $ek[\Lambda]$

$\textsc{Enc}(\Lambda, m_0, m_1, \ell)$   // $\Lambda, m_0, m_1, \ell \in \{0,1\}^*$
If $dk[\Lambda] = \bot$ or $|m_0| \neq |m_1|$ then return $\bot$
$\mathsf{noch} \leftarrow (\mathbf{ch}[\Lambda][s[\Lambda]+1] = \text{“forbidden”})$
$t \leftarrow (\mathsf{noch} \text{ or } \mathsf{exp}[\Lambda])$
If $m_0 \neq m_1$ and $t$ then return $\bot$
$s[\Lambda] \leftarrow s[\Lambda] + 1$
$(ek[\Lambda], c) \leftarrow \mathsf{PKE.Enc}(ek[\Lambda], \ell, m_b; z[\Lambda])$
$z[\Lambda] \leftarrow\!\!\$\ \mathsf{PKE.EncRS}$ ; $\mathsf{nextop} \leftarrow \bot$
$\mathbf{ctable}[\Lambda][s[\Lambda]] \leftarrow (c, \ell)$
If $m_0 \neq m_1$ then $\mathbf{ch}[\Lambda][s[\Lambda]] \leftarrow$ “done”
Return $(ek[\Lambda], c)$

$\textsc{Dec}(\Lambda, c, \ell)$   // $\Lambda, c, \ell \in \{0,1\}^*$
If $dk[\Lambda] = \bot$ or $\mathsf{nextop} \neq \bot$ then return $\bot$
$r[\Lambda] \leftarrow r[\Lambda] + 1$
$(dk[\Lambda], m) \leftarrow\!\!\$\ \mathsf{PKE.Dec}(dk[\Lambda], \ell, c)$
If $(c, \ell) \neq \mathbf{ctable}[\Lambda][r[\Lambda]]$ or $\mathsf{restricted}[\Lambda]$ then
    $\mathsf{restricted}[\Lambda] \leftarrow \mathsf{true}$
    Return $m$
Return $\bot$

$\textsc{ExpRand}(\Lambda)$   // $\Lambda \in \{0,1\}^*$
If $dk[\Lambda] = \bot$ or $\mathsf{nextop} \neq \bot$ then return $\bot$
$\mathbf{ch}[\Lambda][s[\Lambda]+1] \leftarrow$ “forbidden”
$\mathsf{nextop} \leftarrow$ “encrypt” ; Return $z[\Lambda]$

$\textsc{ExpDk}(\Lambda)$   // $\Lambda \in \{0,1\}^*$
If $dk[\Lambda] = \bot$ or $\mathsf{nextop} \neq \bot$ then return $\bot$
If $\mathsf{restricted}[\Lambda]$ then return $dk[\Lambda]$
If $\exists i \in (r[\Lambda], s[\Lambda]]$ s.t. $\mathbf{ch}[\Lambda][i] =$ “done” then
    Return $\bot$
$\mathsf{exp}[\Lambda] \leftarrow \mathsf{true}$ ; Return $dk[\Lambda]$
```

Fig. 3. Games defining signature unforgeability under exposures of key-updatable digital signature scheme DS, and ciphertext indistinguishability under exposures of key-updating public-key encryption scheme PKE.

Signature uniqueness. We will be interested in schemes for which there is only a single signature that will be accepted for any message m and any sequence of updates $\boldsymbol{\Delta}$. Consider game UNIQ of Fig. 1, associated to a key-updatable digital signature scheme DS and an adversary $\mathcal{B}_{\mathsf{DS}}$. The adversary $\mathcal{B}_{\mathsf{DS}}$ can call the oracle NewUser arbitrarily many times with a user identifier Λ and be given the randomness used to generate the keys of Λ. The adversary ultimately outputs a user id Λ, message m, signatures σ_1, σ_2, and key update vector $\boldsymbol{\Delta}$. It wins if the signatures are distinct and both verify for m when the verification key of Λ is updated with $\boldsymbol{\Delta}$. The advantage of $\mathcal{B}_{\mathsf{DS}}$ in breaking the UNIQ security of DS is defined by $\mathsf{Adv}^{\mathsf{uniq}}_{\mathsf{DS}}(\mathcal{B}_{\mathsf{DS}}) = \Pr[\mathrm{UNIQ}^{\mathcal{B}_{\mathsf{DS}}}_{\mathsf{DS}}]$.

Signature unforgeability under exposures. Our main security notion for signatures asks that the adversary not be able to create signatures for any key update vector $\boldsymbol{\Delta}$ unless it was given a signature for that key update vector or given the signing key such that the vector of strings it had been updated with was a prefix of $\boldsymbol{\Delta}$. Consider game UFEXP of Fig. 3, associated to a key-updatable digital signature scheme DS and an adversary $\mathcal{A}_{\mathsf{DS}}$.

The adversary $\mathcal{A}_{\mathsf{DS}}$ can call the oracle NewUser arbitrarily many times for any user identifier Λ and be given the verification key for that user. Then it can interact with user Λ via three different oracles. Via calls to Upd with a string Δ it requests that the signing key for the specified user be updated with Δ. Via calls to Sign with message m it asks for a signature of m using the signing key for the specified user. When it does so the signing key is erased so it can no longer interact with that user and $\boldsymbol{\Delta}^*[\Lambda]$ is used to store the vector of strings the key was updated with.[1] Via calls to Exp it can ask to be given the current signing key of the specified user. When it does so $\boldsymbol{\Delta}'[\Lambda]$ is used to store the vector of strings the key was updated with.

At the end of the game the adversary outputs a user id Λ, signature σ, message m, and key update vector $\boldsymbol{\Delta}$. The adversary has cheated if it previously received σ as the result of calling Sign(Λ, m) and $\boldsymbol{\Delta} = \boldsymbol{\Delta}^*[\Lambda]$, or if it exposed the signing key of Λ and $\boldsymbol{\Delta}'[\Lambda]$ is a prefix of $\boldsymbol{\Delta}$. It wins if it has not cheated and the signature it output verifies for m when the verification key of Λ is updated with $\boldsymbol{\Delta}$. The advantage of $\mathcal{A}_{\mathsf{DS}}$ in breaking the UFEXP security of DS is defined by $\mathsf{Adv}^{\mathsf{ufexp}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{DS}}) = \Pr[\mathrm{UFEXP}^{\mathcal{A}_{\mathsf{DS}}}_{\mathsf{DS}}]$.

Construction. In [27] we use a forward secure [5] key-evolving signature scheme with unique signatures to construct a signature scheme secure with respect to both of the above definitions. Roughly, a key-evolving signature scheme is like a key-updatable digital signature scheme that can only update with $\Delta = \varepsilon$. In order to enable updates with respect to arbitrary key update information, we

[1] We are thus defining security for a *one-time* signature scheme, because a particular key will only be used for one signature. This is all we require for our application, but the definition and construction we provide could easily be extended to allow multiple signatures if desired.

sign each update string with the current key prior to evolving the key, and then include these intermediate signatures with our final signature.

3.2 Key-updating public-key encryption schemes

We start by formally defining the syntax and correctness of a key-updating public-key encryption. Then we specify a security definition for it. We will briefly sketch how to construct such a scheme, but leave the details to [27]. We consider public-key encryption with labels as introduced by Shoup [40].

Syntax and correctness. A key-updating public-key encryption scheme PKE specifies algorithms $\mathsf{PKE.Kg}$, $\mathsf{PKE.Enc}$, $\mathsf{PKE.Dec}$. Associated to PKE is a key generation randomness space $\mathsf{PKE.KgRS}$ and encryption randomness space $\mathsf{PKE.EncRS}$. Key generation algorithm $\mathsf{PKE.Kg}$ takes randomness $z \in \mathsf{PKE.KgRS}$ to return an encryption key ek and a decryption key dk, denoted by $(ek, dk) \leftarrow \mathsf{PKE.Kg}(z)$. Encryption algorithm $\mathsf{PKE.Enc}$ takes ek, a label $\ell \in \{0,1\}^*$, a message $m \in \{0,1\}^*$ and randomness $z \in \mathsf{PKE.EncRS}$ to return a new encryption key ek and a ciphertext c, denoted by $(ek, c) \leftarrow \mathsf{PKE.Enc}(ek, \ell, m; z)$. Decryption algorithm $\mathsf{PKE.Dec}$ takes dk, ℓ, c to return a new decryption key dk and a message $m \in \{0,1\}^*$, denoted by $(dk, m) \leftarrow\!\!\$\; \mathsf{PKE.Dec}(dk, \ell, c)$.

The correctness condition requires that ciphertexts decrypt correctly as long as they are received in the same order they were created and with the same labels. To formalize this, consider game PKECORR of Fig. 2, associated to a key-updating public-key encryption scheme PKE and an adversary $\mathcal{C}$. The advantage of an adversary $\mathcal{C}$ against the correctness of PKE is given by $\mathsf{Adv}^{\mathsf{pkecorr}}_{\mathsf{PKE}}(\mathcal{C}) = \Pr[\mathrm{PKECORR}^{\mathcal{C}}_{\mathsf{PKE}}]$. Correctness requires that $\mathsf{Adv}^{\mathsf{pkecorr}}_{\mathsf{PKE}}(\mathcal{C}) = 0$ for all (even computationally unbounded) adversaries $\mathcal{C}$. See Section 4 for discussion on game-based definitions of correctness.

Define the min-entropy of algorithms $\mathsf{PKE.Kg}$ and $\mathsf{PKE.Enc}$ as $\mathrm{H}_\infty(\mathsf{PKE.Kg})$ and $\mathrm{H}_\infty(\mathsf{PKE.Enc})$, respectively, defined as follows:

$$2^{-\mathrm{H}_\infty(\mathsf{PKE.Kg})} = \max_{ek} \Pr\left[ek^* = ek : (ek^*, dk^*) \leftarrow\!\!\$\; \mathsf{PKE.Kg}\right],$$

$$2^{-\mathrm{H}_\infty(\mathsf{PKE.Enc})} = \max_{ek,\ell,m,c} \Pr\left[c^* = c : (ek^*, c^*) \leftarrow\!\!\$\; \mathsf{PKE.Enc}(ek, \ell, m)\right] .$$

The probability is defined over the random coins used by $\mathsf{PKE.Kg}$ and $\mathsf{PKE.Enc}$, respectively. Note that min-entropy does not depend on the output values dk^* (in the former case) and ek^* (in the latter case).

Ciphertext indistinguishability under exposures. Consider game INDEXP of Fig. 3, associated to a key-updating public-key encryption scheme PKE and an adversary $\mathcal{A}_{\mathsf{PKE}}$. Roughly, it requires that PKE maintain CCA security [3] even if $\mathcal{A}_{\mathsf{PKE}}$ is given the decryption key (as long as that decryption key is no longer able to decrypt any challenge ciphertexts).

The adversary $\mathcal{A}_{\mathsf{PKE}}$ can call the oracle NEWUSER arbitrarily many times with a user identifier Λ and be given the encryption key of that user. Then it can interact with user Λ via four oracles. Via calls to ENC with messages m_0, m_1 and label ℓ it requests that one of these messages be encrypted using the specified label (which message is encrypted depends on the secret bit b). It will be given back the new encryption key and the produced ciphertext. If $m_0 \neq m_1$ we remember that a challenge query was done.

Via calls to DEC with ciphertext c and ℓ it requests that the ciphertext be decrypted with the specified label. Adversary $\mathcal{A}_{\mathsf{PKE}}$ will only be given the result of this decryption if the pair (c, ℓ) was not obtained from a call to ENC. Once the adversary queries such pair, the user Λ becomes "restricted" and the oracle will return the true decryption of all future ciphertexts for this user.

Via calls to EXPRAND it asks to be given the next randomness that will be used for encryption. This represents the adversary exposing the randomness while the encryption is taking place so we require that after a call to EXPRAND the adversary immediately makes the corresponding call to ENC. During this call challenges are forbidden so it must choose $m_0 = m_1$.

Via calls to EXPDK it asks to be given the current decryption key of the user. It may not do so if a challenge query was done but the user has not decrypted the corresponding ciphertext yet (unless the user is restricted). Otherwise the decryption key is returned and the user is considered to be exposed. Once a user is exposed challenges are not allowed so for all future calls to ENC the adversary required to choose $m_0 = m_1$.

At the end of the game the adversary outputs a bit b' representing its guess of the secret bit b. The advantage of $\mathcal{A}_{\mathsf{PKE}}$ in breaking the INDEXP security of PKE is defined as $\mathsf{Adv}^{\mathsf{indexp}}_{\mathsf{PKE}}(\mathcal{A}_{\mathsf{PKE}}) = 2\Pr[\mathrm{INDEXP}^{\mathcal{A}_{\mathsf{PKE}}}_{\mathsf{PKE}}] - 1$.

Many of the variables used to track the behavior of the adversary in INDEXP are analogous to variables we use and discuss in detail in Section 5 when defining security of a channel. The reader interested in understand the pseudocode of INDEXP in detail is encouraged to read that section first.

Construction. In [27] we use a hierarchical identity-based encryption (HIBE) scheme to construct a secure key-updating encryption scheme. Roughly, a HIBE assigns a decryption key to any identity (vector of strings). A decryption key for an identity $\boldsymbol{I}$ can be used to create decryption keys for an identity of which $\boldsymbol{I}$ is a prefix. Security requires that the adversary be unable to learn about encrypted messages encrypted to an identity $\boldsymbol{I}$ even if given the decryption key for many identities as long as none of them were prefixes of $\boldsymbol{I}$. To create a key-updating encryption scheme we use the vector of ciphertexts and labels a user has received so far as the identity. The security of this scheme then follows from the security of the underlying HIBE in a fairly straightforward manner.

4 Bidirectional cryptographic channels

In this section we formally define the syntax and correctness of bidirectional cryptographic channels. Our notion of bidirectional channels will closely match that of Marson and Poettering [30]. Compared to their definition, we allow the receiving algorithm to be randomized and provide an alternative correctness condition. We argue that the new correctness condition is more appropriate for our desired use case of secure messaging. Henceforth, we will omit the adjective "bidirectional" and refer simply to channels.

Syntax of channel. A channel provides a method for two users to exchange messages in an arbitrary order. We will refer to the two users of a channel as the initiator $\mathcal{I}$ and the receiver $\mathcal{R}$. There will be no formal distinction between the two users, but when specifying attacks we follow the convention of having $\mathcal{I}$ send a ciphertext first. We will use u as a variable to represent an arbitrary user and $\overline{u}$ to represent the other user. More formally, when $u \in \{\mathcal{I}, \mathcal{R}\}$ we let $\overline{u}$ denote the sole element of $\{\mathcal{I}, \mathcal{R}\} \setminus \{u\}$.

A channel Ch specifies algorithms $\mathsf{Ch.Init}$, $\mathsf{Ch.Send}$, and $\mathsf{Ch.Recv}$. Initialization algorithm $\mathsf{Ch.Init}$ returns initial states $st_\mathcal{I} \in \{0,1\}^*$ and $st_\mathcal{R} \in \{0,1\}^*$, where $st_\mathcal{I}$ is $\mathcal{I}$'s state and $st_\mathcal{R}$ is $\mathcal{R}$'s state. We write $(st_\mathcal{I}, st_\mathcal{R}) \leftarrow\!\!\$\ \mathsf{Ch.Init}$. Sending algorithm $\mathsf{Ch.Send}$ takes state $st_u \in \{0,1\}^*$, associated data $ad \in \{0,1\}^*$, and message $m \in \{0,1\}^*$ to return updated state $st_u \in \{0,1\}^*$ and a ciphertext $c \in \{0,1\}^*$. We write $(st_u, c) \leftarrow\!\!\$\ \mathsf{Ch.Send}(st_u, ad, m)$. Receiving algorithm takes state $st_u \in \{0,1\}^*$, associated data $ad \in \{0,1\}^*$, and ciphertext $c \in \{0,1\}^*$ to return updated state $st_u \in \{0,1\}^* \cup \{\bot\}$ and message $m \in \{0,1\}^* \cup \{\bot\}$. We write $(st_u, m) \leftarrow\!\!\$\ \mathsf{Ch.Recv}(st_u, ad, c)$, where $m = \bot$ represents a rejection of ciphertext c and $st_u = \bot$ represents the channel being permanently shut down from the perspective of u (recall our convention regarding $\bot$ as input to an algorithm). One notion of correctness we discuss will require that $st_u = \bot$ whenever $m = \bot$. The other will require that st_u not be changed from its input value when $m = \bot$.

We let $\mathsf{Ch.InitRS}$, $\mathsf{Ch.SendRS}$, and $\mathsf{Ch.RecvRS}$ denote the sets of possible random coins for $\mathsf{Ch.Init}$, $\mathsf{Ch.Send}$, and $\mathsf{Ch.Recv}$, respectively. Note that for full generality we allow $\mathsf{Ch.Recv}$ to be randomized. Prior work commonly requires this algorithm to be deterministic.

Correctness of channel. In Fig. 4 we provide two games, defining two alternative correctness requirements for a cryptographic channel. Lines labelled with the name of a game are included only in that game. The games differ in whether the adversary is given access to an oracle ROBUST or to an oracle REJECT. Game CORR uses the former, whereas game CORR⊥ uses the latter. The advantage of an adversary $\mathcal{C}$ against the correctness of channel Ch is given by $\mathsf{Adv}^{\mathsf{corr}}_{\mathsf{Ch}}(\mathcal{C}) = \Pr[\mathrm{CORR}^{\mathcal{C}}_{\mathsf{Ch}}]$ in one case, and $\mathsf{Adv}^{\mathsf{corr}\bot}_{\mathsf{Ch}}(\mathcal{C}) = \Pr[\mathrm{CORR}\bot^{\mathcal{C}}_{\mathsf{Ch}}]$ in the other case. Correctness with respect to either notion requires that the advantage is equal 0 for all (even computationally unbounded) adversaries $\mathcal{C}$.

```
Games $\text{CORR}^{\mathcal{C}}_{\mathsf{Ch}}, \text{CORR}\bot^{\mathcal{C}}_{\mathsf{Ch}}$
$s_{\mathcal{I}} \leftarrow r_{\mathcal{I}} \leftarrow s_{\mathcal{R}} \leftarrow r_{\mathcal{R}} \leftarrow 0$ ; $\nu \leftarrow\$ \mathsf{Ch.InitRS}$ ; $(st_{\mathcal{I}}, st_{\mathcal{R}}) \leftarrow \mathsf{Ch.Init}(\nu)$
$\mathcal{C}^{\text{Send,Recv,Robust}}(\nu)$  // $\text{CORR}^{\mathcal{C}}_{\mathsf{Ch}}$
$\mathcal{C}^{\text{Send,Recv,Reject}}(\nu)$  // $\text{CORR}\bot^{\mathcal{C}}_{\mathsf{Ch}}$
Return $\mathsf{bad}$

$\text{Send}(u, ad, m)$  // $u \in \{\mathcal{I}, \mathcal{R}\}, (ad, m) \in (\{0,1\}^*)^2$
$s_u \leftarrow s_u + 1$ ; $z \leftarrow\$ \mathsf{Ch.SendRS}$ ; $(st_u, c) \leftarrow \mathsf{Ch.Send}(st_u, ad, m; z)$
$\mathbf{ctable}_{\overline{u}}[s_u] \leftarrow (c, ad)$ ; $\mathbf{mtable}_{\overline{u}}[s_u] \leftarrow m$ ; Return $z$

$\text{Recv}(u)$  // $u \in \{\mathcal{I}, \mathcal{R}\}$
If $\mathbf{ctable}_u[r_u + 1] = \bot$ then return $\bot$
$r_u \leftarrow r_u + 1$ ; $(c, ad) \leftarrow \mathbf{ctable}_u[r_u]$
$\eta \leftarrow\$ \mathsf{Ch.RecvRS}$ ; $(st_u, m) \leftarrow \mathsf{Ch.Recv}(st_u, ad, c; \eta)$
If $m \neq \mathbf{mtable}_u[r_u]$ then $\mathsf{bad} \leftarrow \mathsf{true}$
Return $\eta$

$\text{Robust}(st, ad, c)$  // $(st, ad, c) \in (\{0,1\}^*)^3$
$(st', m) \leftarrow\$ \mathsf{Ch.Recv}(st, ad, c)$
If $m = \bot$ and $st' \neq st$ then $\mathsf{bad} \leftarrow \mathsf{true}$

$\text{Reject}(st, ad, c)$  // $(st, ad, c) \in (\{0,1\}^*)^3$
$(st', m) \leftarrow\$ \mathsf{Ch.Recv}(st, ad, c)$
If $m = \bot$ and $st' \neq \bot$ then $\mathsf{bad} \leftarrow \mathsf{true}$
```

Fig. 4. Games defining correctness of channel Ch. Lines labelled with the name of a game are included only in that game. CORR requires that Ch be robust when given an incorrect ciphertext via oracle Robust. CORR⊥ requires that Ch permanently returns ⊥ when given an incorrect ciphertext via oracle Reject.

Our use of games to define correctness conditions follows the work of Marson and Poettering [30] and Bellare et. al. [7]. By considering unbounded adversaries and requiring an advantage of 0 we capture a typical information-theoretic perfect correctness requirement without having to explicitly quantify over sequences of actions. In this work we require only the perfect correctness because it is achieved by our scheme; however, it would be possible to capture computational correctness by considering a restricted class of adversaries.

Both games require that ciphertexts sent by any user are always decrypted to the correct message by the other user. This is modeled by providing adversary $\mathcal{C}$ with access to oracles Send and Recv. We assume that messages from u to $\overline{u}$ are received in the same order they were sent, and likewise that messages from $\overline{u}$ to u are also received in the correct order (regardless Aof how they are interwoven on both sides, since ciphertexts are being sent in both directions).

The games differ in how the channel is required to behave in the case that a ciphertext is rejected. Game CORR (using oracle Robust) requires that the state of the user not be changed so that the channel can continue to be used. Game CORR⊥ (using oracle Reject) requires that the state of the user is set to $\perp$. According to our conventions about the behavior of algorithms given $\perp$ as input (see Section 2), the channel will then refuse to perform any further actions by setting all subsequent outputs to $\perp$. We emphasize that the adversary specifies all inputs to Ch.Recv when making calls to Robust and Reject, so the behavior of those oracles is not related to the behavior of the other two oracles for which the game maintains the state of both users.

Comparison of correctness notions. The correctness required by CORR$\perp$ is identical to that of Marson and Poettering [30]. The CORR notion of correctness instead uses a form of robustness analogous to that of [7]. In [27] we discuss how these correctness notions have different implications for the *security* of the channel. It is trivial to convert a CORR-correct channel to a CORR$\perp$-correct channel and vice versa. Thus we will, without loss of generality, only provide a scheme achieving CORR-correctness.

5 Security notion for channels

In this section we will define what it means for a channel to be secure in the presence of a strong attacker that can steal the secrets of either party in the communication. Our goal is to give the strongest possible notion of security in this setting, encompassing both the privacy of messages and the integrity of ciphertexts. We take a fine-grained look at what attacks are possible and require that a channel be secure against all attacks that are not syntactically inherent in the definition of a channel.

To introduce our security notion we will first describe a simple interface of how the adversary is allowed to interact with the channel. Then we show attacks that would break the security of *any* channel using this interface. Our final security notion will be created by adding checks to the interface that prevents adversary from performing any sequence of actions that leads to these unpreventable breaches of security. We introduce only the minimal necessary restrictions preventing the attacks, making sure that we allow *all* adversaries that do not trivially break the security as per above.

5.1 Channel interface game

Consider game INTER in Fig. 5. It defines the interface between an adversary $\mathcal{D}$ and a channel Ch. A secret bit b is chosen at random and the adversary's goal is to guess this bit given access to a left-or-right sending oracle, real-or-$\perp$ receiving oracle, and an exposure oracle. The sending oracle takes as input a user $u \in \{\mathcal{I}, \mathcal{R}\}$, two messages $m_0, m_1 \in \{0,1\}^*$, and associated data ad. Then it returns the encryption of m_b with ad by user u. The receiving oracle Recv takes as input a user u, a ciphertext c, and associated data ad. It has user u decrypt this ciphertext using ad, and proceeds as follows. If $b = 0$ holds (along with another condition we discuss momentarily) then it returns the valid decryption of this

```
Game $\text{INTER}_{\mathsf{Ch}}^{\mathcal{D}}$
$b \leftarrow\$ \{0,1\}$
$s_{\mathcal{I}} \leftarrow r_{\mathcal{I}} \leftarrow s_{\mathcal{R}} \leftarrow r_{\mathcal{R}} \leftarrow 0$
$(st_{\mathcal{I}}, st_{\mathcal{R}}) \leftarrow\$ \mathsf{Ch.Init}$
$(z_{\mathcal{I}}, z_{\mathcal{R}}) \leftarrow\$ (\mathsf{Ch.SendRS})^2$
$(\eta_{\mathcal{I}}, \eta_{\mathcal{R}}) \leftarrow\$ (\mathsf{Ch.RecvRS})^2$
$b' \leftarrow\$ \mathcal{D}^{\textsc{Send},\textsc{Recv},\textsc{Exp}}$
Return $(b' = b)$

$\textsc{Send}(u, m_0, m_1, ad)$
// $u \in \{\mathcal{I},\mathcal{R}\}, (m_0, m_1, ad) \in (\{0,1\}^*)^3$
If $\mathsf{nextop} \neq (u, \text{“send”})$
    and $\mathsf{nextop} \neq \bot$ then return $\bot$
If $|m_0| \neq |m_1|$ then return $\bot$
$(st_u, c) \leftarrow \mathsf{Ch.Send}(st_u, ad, m_b; z_u)$
$\mathsf{nextop} \leftarrow \bot$
$s_u \leftarrow s_u + 1$ ; $z_u \leftarrow\$ \mathsf{Ch.SendRS}$
$\mathbf{ctable}_{\overline{u}}[s_u] \leftarrow (c, ad)$
Return $c$
```

```
$\textsc{Recv}(u, c, ad)$
// $u \in \{\mathcal{I},\mathcal{R}\}, (c, ad) \in (\{0,1\}^*)^2$
If $\mathsf{nextop} \neq (u, \text{“recv”})$
    and $\mathsf{nextop} \neq \bot$ then return $\bot$
$(st_u, m) \leftarrow \mathsf{Ch.Recv}(st_u, ad, c; \eta_u)$
$\mathsf{nextop} \leftarrow \bot$ ; $\eta_u \leftarrow\$ \mathsf{Ch.RecvRS}$
If $m \neq \bot$ then $r_u \leftarrow r_u + 1$
If $b = 0$ and $(c, ad) \neq \mathbf{ctable}_u[r_u]$ then
    Return $m$
Return $\bot$

$\textsc{Exp}(u, \mathsf{rand})$
// $u \in \{\mathcal{I},\mathcal{R}\}, \mathsf{rand} \in \{\varepsilon, \text{“send”}, \text{“recv”}\}$
If $\mathsf{nextop} \neq \bot$ then return $\bot$
$(z, \eta) \leftarrow (\varepsilon, \varepsilon)$
If $\mathsf{rand} = \text{“send”}$ then
    $\mathsf{nextop} \leftarrow (u, \text{“send”})$ ; $z \leftarrow z_u$
Else if $\mathsf{rand} = \text{“recv”}$ then
    $\mathsf{nextop} \leftarrow (u, \text{“recv”})$ ; $\eta \leftarrow \eta_u$
Return $(st_u, z, \eta)$
```

Fig. 5. Game defining interface between adversary $\mathcal{D}$ and channel Ch.

ciphertext; otherwise it returns $\bot$. The exposure oracle $\textsc{Exp}$ takes as input a user u, and a flag rand. It returns user's state st_u, and it might return random coins that will be used the next time this user runs algorithms $\mathsf{Ch.Send}$ or $\mathsf{Ch.Recv}$ (depending on the value of rand, which we discuss below). The advantage of adversary $\mathcal{D}$ against channel Ch is defined by $\mathsf{Adv}_{\mathsf{Ch}}^{\mathsf{inter}}(\mathcal{D}) = 2\Pr[\text{INTER}_{\mathsf{Ch}}^{\mathcal{D}}] - 1$.

This interface gives the adversary full control over the communication between the two users of the channel. It may modify, reorder, or block any communication as it sees fit. The adversary is able to exfiltrate the secret state of either party at any time.

Let us consider the different cases of how a user's secrets might be exposed. They could be exposed while the user is in the middle of performing a $\mathsf{Ch.Send}$ operation, in the middle of performing a $\mathsf{Ch.Recv}$ operation, or when the user is idle (i.e. not in the middle of performing $\mathsf{Ch.Send}$ or $\mathsf{Ch.Recv}$). In the last case we expect the adversary to learn the user's state st_u, but nothing else. If the adversary is exposing the user during an operation, they would potentially learn the state before the operation, any secrets computed during the operation, and the state after the operation. We capture this by leaking the state from before the operation along with the randomness that will be used when the adversary makes its next query to $\textsc{Send}$ or $\textsc{Recv}$. This allows the adversary to compute the next state as well. The three possible values of rand are rand = “send” for the first possibility, rand = “recv” for the second possibility, and $\mathsf{rand} = \varepsilon$

for the third. These exposures represent what the adversary is learning while a particular operation is occurring, so we require (via nextop) that after such an exposure it immediately makes the corresponding oracle query. Without the use of the exposure oracle the game specified by this interface would essentially be equivalent to the combination of the integrity and confidentiality security notions defined by Marson and Poettering [30] in the all-in-one definition style of Rogaway and Shrimpton [37].

The interface game already includes some standard checks. First, we require that on any query (u, m_0, m_1, ad) to $\textsc{Send}$ the adversary must provide equal length messages. If the adversary does not do so (i.e. $|m_0| \neq |m_1|$) then $\textsc{Send}$ returns $\perp$ immediately. This prevents the inherent attack where an adversary could distinguish between the two values of b by asking for encryptions of different length messages and checking the length of the output ciphertext. Adversary $\mathcal{D}_1$ in Fig. 6 does just that and would achieve $\mathsf{Adv}^{\mathsf{inter}}_{\mathsf{Ch}}(\mathcal{D}_1) > 1/2$ against any channel Ch if not for that check.

Second, we want to prevent $\textsc{Recv}$ from decrypting ciphertexts that are simply forwarded to it from $\textsc{Send}$. So for each user u we keep track of counters s_u and r_u that track how many messages that user has sent and received. Then at the end of a $\textsc{Send}$ call to u the ciphertext-associated data pair (c, ad) is stored in the table $\mathbf{ctable}_{\overline{u}}$ with index s_u. When $\textsc{Recv}$ is called for user $\overline{u}$ it will compare the pair (c, ad) against $\mathbf{ctable}_{\overline{u}}[r_{\overline{u}}]$ and if the pair matches return $\perp$ regardless of the value of the secret bit. If we did not do this check then for any channel Ch the adversary $\mathcal{D}_2$ shown in Fig. 6 would achieve $\mathsf{Adv}^{\mathsf{inter}}_{\mathsf{Ch}}(\mathcal{D}_2) = 1$.

We now specify several efficient adversaries that will have high advantage for *any* choice of Ch. For concreteness we always have our adversaries immediately start the actions required to perform the attacks, but all of the attacks would still work if the adversary had performed a number of unrelated procedure calls first. Associated data will never be important for our attacks so we will always set it to ε. We will typically set $m_0 = 0$ and $m_1 = 1$. For the following we let Ch be any channel and consider the adversaries shown in Fig. 6.

Trivial Forgery. If the adversary exposes the secrets of u it will be able to forge a ciphertext that $\overline{u}$ would accept at least until the future point in time when $\overline{u}$ has received the ciphertext that u creates next. For a simple example of this consider the third adversary, $\mathcal{D}_3$. It exposes the secrets of user $\mathcal{I}$, then uses them to perform its own $\mathsf{Ch.Send}$ computation locally, and sends the resulting ciphertext to $\mathcal{R}$. Clearly this ciphertext will always decrypt to a non-$\perp$ value so the adversary can trivially determine the value of b and achieve $\mathsf{Adv}^{\mathsf{inter}}_{\mathsf{Ch}}(\mathcal{D}_3) = 1$.

After an adversary has done the above to trivially send a forgery to $\overline{u}$ it can easily perform further attacks on both the integrity and authenticity of the channel. These are shown by adversaries $\mathcal{D}_{3.1}$ and $\mathcal{D}_{3.2}$. The first displays the fact that the attacker can easily send further forgeries to $\overline{u}$. The second displays the fact that the attacker can now easily decrypt any messages sent by $\overline{u}$. We have $\mathsf{Adv}^{\mathsf{inter}}_{\mathsf{Ch}}(\mathcal{D}_{3.1}) = 1$ and $\mathsf{Adv}^{\mathsf{inter}}_{\mathsf{Ch}}(\mathcal{D}_{3.2}) = 1$.

```
Adversary $\mathcal{D}_1^{\textsc{Send},\textsc{Recv},\textsc{Exp}}$
$(st, z, \eta) \leftarrow \textsc{Exp}(\mathcal{I}, \varepsilon)$
$n \leftarrow \max_{c \in [\mathsf{Ch.Send}(st,\varepsilon,1)]} |c|$
$m \leftarrow_{\$} \{0,1\}^{n+2}$
$c \leftarrow \textsc{Send}(\mathcal{I}, m, 1, \varepsilon)$
If $|c| \leq n$ then return 1
Return 0
```

```
Adversary $\mathcal{D}_2^{\textsc{Send},\textsc{Recv},\textsc{Exp}}$
$c \leftarrow \textsc{Send}(\mathcal{I}, 1, 1, \varepsilon)$
$m \leftarrow \textsc{Recv}(\mathcal{R}, c, \varepsilon)$
If $m = \perp$ then return 1
Return 0
```

```
Adversary $\mathcal{D}_3^{\textsc{Send},\textsc{Recv},\textsc{Exp}}$
$(st, z, \eta) \leftarrow \textsc{Exp}(\mathcal{I}, \varepsilon)$
$(st, c) \leftarrow_{\$} \mathsf{Ch.Send}(st, \varepsilon, 1)$
$m \leftarrow \textsc{Recv}(\mathcal{R}, c, \varepsilon)$
If $m = \perp$ then return 1
Return 0
```

```
Adversary $\mathcal{D}_{3.1}^{\textsc{Send},\textsc{Recv},\textsc{Exp}}$
$(st, z, \eta) \leftarrow \textsc{Exp}(\mathcal{I}, \varepsilon)$
$(st, c) \leftarrow_{\$} \mathsf{Ch.Send}(st, \varepsilon, 1)$
$m \leftarrow \textsc{Recv}(\mathcal{R}, c, \varepsilon)$
$(st, c) \leftarrow_{\$} \mathsf{Ch.Send}(st, \varepsilon, 1)$
$m \leftarrow \textsc{Recv}(\mathcal{R}, c, \varepsilon)$
If $m = \perp$ then return 1
Return 0
```

```
Adversary $\mathcal{D}_{3.2}^{\textsc{Send},\textsc{Recv},\textsc{Exp}}$
$(st, z, \eta) \leftarrow \textsc{Exp}(\mathcal{I}, \varepsilon)$
$(st, c) \leftarrow_{\$} \mathsf{Ch.Send}(st, \varepsilon, 1)$
$m \leftarrow \textsc{Recv}(\mathcal{R}, c, \varepsilon)$
$c \leftarrow \textsc{Send}(\mathcal{R}, 0, 1, \varepsilon)$
$(st, m) \leftarrow_{\$} \mathsf{Ch.Recv}(st, \varepsilon, c)$
If $m = 1$ then return 1
Return 0
```

```
Adversary $\mathcal{D}_4^{\textsc{Send},\textsc{Recv},\textsc{Exp}}$
$c \leftarrow \textsc{Send}(\mathcal{I}, 0, 1, \varepsilon)$
$(st, z, \eta) \leftarrow \textsc{Exp}(\mathcal{R}, \varepsilon)$
$(st, m) \leftarrow_{\$} \mathsf{Ch.Recv}(st, \varepsilon, c)$
If $m = 1$ then return 1
Return 0
```

```
Adversary $\mathcal{D}_5^{\textsc{Send},\textsc{Recv},\textsc{Exp}}$
$(st, z, \eta) \leftarrow \textsc{Exp}(\mathcal{R}, \varepsilon)$
$c \leftarrow \textsc{Send}(\mathcal{I}, 0, 1, \varepsilon)$
$(st, m) \leftarrow_{\$} \mathsf{Ch.Recv}(st, \varepsilon, c)$
If $m = 1$ then return 1
Return 0
```

```
Adversary $\mathcal{D}_6^{\textsc{Send},\textsc{Recv},\textsc{Exp}}$
$(st, z, \eta) \leftarrow \textsc{Exp}(\mathcal{I}, \text{“send”})$
$(st, c) \leftarrow \mathsf{Ch.Send}(st, \varepsilon, 1; z)$
$c' \leftarrow \textsc{Send}(\mathcal{I}, 0, 1, \varepsilon)$
If $c' = c$ then return 1
Return 0
```

Fig. 6. Generic attacks against any channel Ch with interface INTER.

Trivial Challenges. If the adversary exposes the secrets of u it will necessarily be able to decrypt any ciphertexts already encrypted by $\overline{u}$ that have not already been received by u. Consider the adversary $\mathcal{D}_4$. It determines what message was encrypted by user $\mathcal{I}$ by exposing the state of $\mathcal{R}$, and uses that to run Ch.Recv. We have $\mathsf{Adv}_{\mathsf{Ch}}^{\mathsf{inter}}(\mathcal{D}_4) = 1$.

Similarly, if the adversary exposes the secrets of u it will necessarily be able to decrypt any future ciphertexts encrypted by $\overline{u}$, until $\overline{u}$ receives the ciphertext that u creates next. Consider the adversary $\mathcal{D}_5$. It is essentially the identical to adversary $\mathcal{D}_4$, except it reverses the order of the calls made to Send and Exp. We have $\mathsf{Adv}_{\mathsf{Ch}}^{\mathsf{inter}}(\mathcal{D}_5) = 1$.

Exposing Randomness. If an adversary exposes user u with rand = “send” then it is able to compute the next state of u by running Ch.Send locally with the same randomness that u will use. So in this case the security game must act as if the adversary exposed both the current and the next state. In particular, the attacks above could only succeed until, first, the exposed user u updated its secrets and, second, user $\overline{u}$ updates its secrets accordingly (which can happen after it receives the next message from u). But if the randomness was exposed, then secrets would need to be updated at least twice until the security is restored.

Exposing user u with rand = “send” additionally allows the attack shown in $\mathcal{D}_6$. The adversary exposes the state and the sending randomness of $\mathcal{I}$, encrypts

1 locally using these exposed values of $\mathcal{I}$, and then calls SEND to get a challenge ciphertext sent by $\mathcal{I}$. The adversary compares whether the two ciphertexts are the same to determine the secret bit. We have $\mathsf{Adv}^{\mathsf{inter}}_{\mathsf{Ch}}(\mathcal{D}_6) = 1$. More broadly, if the adversary exposes the secrets of u with $\mathsf{rand} =$ “send” it will always be able to tell what is the next message encrypted by u.

Exposing with $\mathsf{rand} =$ “recv” does not generically endow the adversary with the ability to do any additional attacks.

5.2 Optimal security of a channel

Our full security game is obtained by adding a minimal amount of code to INTER to disallow the generic attacks just discussed. Consider the game AEAC (authenticated encryption against compromise) shown in Fig. 7. We define the advantage of an adversary $\mathcal{D}$ against channel Ch by $\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{Ch}}(\mathcal{D}) = 2\Pr[\mathrm{AEAC}^{\mathcal{D}}_{\mathsf{Ch}}] - 1$.

We now have a total of eight variables to control the behavior of the adversary and prevent it from abusing trivial attacks. Some of the variables are summarized in Fig. 8. We have already seen s_u, r_u, nextop, and $\mathbf{ctable}_u$ in INTER. The new variables we have added in AEAC are tables $\mathbf{forge}_u$ and $\mathbf{ch}_u$, number $\mathcal{X}_u \in \mathbb{N}$, and flag $\mathsf{restricted}_u \in \{\mathsf{true}, \mathsf{false}\}$. We now discuss the new variables.

The table $\mathbf{forge}_u$ was added to prevent the type of attack shown in $\mathcal{D}_3$. When the adversary calls EXP on user u we set $\mathbf{forge}_{\overline{u}}$ to “trivial” for the indices of ciphertexts for which this adversary is now necessarily able to create forgeries. If the adversary takes advantage of this to send a ciphertext of its own creation to $\overline{u}$ then the flag $\mathsf{restricted}_{\overline{u}}$ will be set, whose effect we will describe momentarily.

The table $\mathbf{ch}_u$ is used to prevent the types of attacks shown by $\mathcal{D}_4$ and $\mathcal{D}_6$. Whenever the adversary makes a valid challenge query[2] to user u we set $\mathbf{ch}_u[s_u]$ to “done”. The game will not allow the adversary to expose $\overline{u}$’s secrets if there are any challenge queries for which u sent a ciphertext that $\overline{u}$ has not received yet. This use of $\mathbf{ch}_u$ prevents an attack like $\mathcal{D}_4$. To prevent an attack like $\mathcal{D}_6$, we set $\mathbf{ch}_u[s_u + 1]$ to “forbidden” whenever the adversary exposes the state and sending randomness of u. This disallows the adversary from doing a challenge query during its next SEND call to u (the call for which the adversary knows the corresponding randomness).

The number $\mathcal{X}_u$ prevents attacks like $\mathcal{D}_5$. When u is exposed $\mathcal{X}_{\overline{u}}$ will be set to a number that is 1 or 2 greater than the current number of ciphertexts u has sent (depending on the value of rand) and challenge queries from $\overline{u}$ will not be allowed until it has received that many ciphertexts. This ensures that the challenge queries from $\overline{u}$ are not issued with respect to exposed keys of u.[3]

Finally the flag $\mathsf{restricted}_u$ serves to both allow and disallow some attacks. The flag is initialized to false. It is set to true when the adversary forges a

[2] We use the term challenge query to refer to a SEND query for which $m_0 \neq m_1$.

[3] The symbol chi is meant to evoke the word “challenge” because it stores the next time the adversary may make a challenge query.

```
Game AEAC$_{\mathsf{Ch}}^{\mathcal{D}}$
$b \leftarrow_{\$} \{0,1\}$ ; $s_{\mathcal{I}} \leftarrow 0$ ; $r_{\mathcal{I}} \leftarrow 0$ ; $s_{\mathcal{R}} \leftarrow 0$ ; $r_{\mathcal{R}} \leftarrow 0$
$\mathsf{restricted}_{\mathcal{I}} \leftarrow \mathsf{false}$ ; $\mathsf{restricted}_{\mathcal{R}} \leftarrow \mathsf{false}$
$\mathbf{forge}_{\mathcal{I}}[0 \ldots \infty] \leftarrow$ “nontrivial” ; $\mathbf{forge}_{\mathcal{R}}[0 \ldots \infty] \leftarrow$ “nontrivial”
$\mathcal{X}_{\mathcal{I}} \leftarrow 0$ ; $\mathcal{X}_{\mathcal{R}} \leftarrow 0$ ; $(st_{\mathcal{I}}, st_{\mathcal{R}}) \leftarrow_{\$} \mathsf{Ch.Init}$
$(z_{\mathcal{I}}, z_{\mathcal{R}}) \leftarrow_{\$} (\mathsf{Ch.SendRS})^2$ ; $(\eta_{\mathcal{I}}, \eta_{\mathcal{R}}) \leftarrow_{\$} (\mathsf{Ch.RecvRS})^2$
$b' \leftarrow_{\$} \mathcal{D}^{\text{SEND},\text{RECV},\text{EXP}}$
Return $(b' = b)$

SEND$(u, m_0, m_1, ad)$  // $u \in \{\mathcal{I}, \mathcal{R}\}, (m_0, m_1, ad) \in (\{0,1\}^*)^3$
If $\mathsf{nextop} \neq (u, \text{“send”})$ and $\mathsf{nextop} \neq \perp$ then return $\perp$
If $|m_0| \neq |m_1|$ then return $\perp$
If $(r_u < \mathcal{X}_u$ or $\mathsf{restricted}_u$ or $\mathbf{ch}_u[s_u + 1] =$ “forbidden”) and $m_0 \neq m_1$ then
  Return $\perp$
$(st_u, c) \leftarrow \mathsf{Ch.Send}(st_u, ad, m_b; z)$
$\mathsf{nextop} \leftarrow \perp$ ; $s_u \leftarrow s_u + 1$ ; $z_u \leftarrow_{\$} \mathsf{Ch.SendRS}$
If not $\mathsf{restricted}_u$ then $\mathbf{ctable}_{\overline{u}}[s_u] \leftarrow (c, ad)$
If $m_0 \neq m_1$ then $\mathbf{ch}_u[s_u] \leftarrow$ “done”
Return $c$

RECV$(u, c, ad)$  // $u \in \{\mathcal{I}, \mathcal{R}\}, (c, ad) \in (\{0,1\}^*)^2$
If $\mathsf{nextop} \neq (u, \text{“recv”})$ and $\mathsf{nextop} \neq \perp$ then return $\perp$
$(st_u, m) \leftarrow \mathsf{Ch.Recv}(st_u, ad, c; \eta_u)$
$\mathsf{nextop} \leftarrow \perp$ ; $\eta_u \leftarrow_{\$} \mathsf{Ch.RecvRS}$
If $m = \perp$ then return $\perp$
$r_u \leftarrow r_u + 1$
If $\mathbf{forge}_u[r_u] =$ “trivial” and $(c, ad) \neq \mathbf{ctable}_u[r_u]$ then
  $\mathsf{restricted}_u \leftarrow \mathsf{true}$
If $\mathsf{restricted}_u$ or $(b = 0$ and $(c, ad) \neq \mathbf{ctable}_u[r_u])$ then
  Return $m$
Return $\perp$

EXP$(u, \mathsf{rand})$  // $u \in \{\mathcal{I}, \mathcal{R}\}, \mathsf{rand} \in \{\varepsilon, \text{“send”}, \text{“recv”}\}$
If $\mathsf{nextop} \neq \perp$ then return $\perp$
If $\mathsf{restricted}_u$ then return $(st_u, z_u, \eta_u)$
If $\exists i \in (r_u, s_{\overline{u}}]$ s.t. $\mathbf{ch}_{\overline{u}}[i] =$ “done” then
  Return $\perp$
$\mathbf{forge}_{\overline{u}}[s_u + 1] \leftarrow$ “trivial” ; $(z, \eta) \leftarrow (\varepsilon, \varepsilon)$ ; $\mathcal{X}_{\overline{u}} \leftarrow s_u + 1$
If $\mathsf{rand} =$ “send” then
  $\mathsf{nextop} \leftarrow (u, \text{“send”})$ ; $z \leftarrow z_u$ ; $\mathcal{X}_{\overline{u}} \leftarrow s_u + 2$
  $\mathbf{forge}_{\overline{u}}[s_u + 2] \leftarrow$ “trivial” ; $\mathbf{ch}_u[s_u + 1] \leftarrow$ “forbidden”
Else if $\mathsf{rand} =$ “recv” then
  $\mathsf{nextop} \leftarrow (u, \text{“recv”})$ ; $\eta \leftarrow \eta_u$
Return $(st_u, z, \eta)$
```

Fig. 7. Game defining AEAC security of channel Ch.

Variable	Set to x when y occurs	Effect
nextop_u	$(u, \text{"send"})$ when z_u is exposed	$-$ u must send next
	$(u, \text{"recv"})$ when η_u is exposed	$-$ u must receive next
$\mathbf{forge}_u$	"trivial" when $\overline{u}$ is exposed	$-$ forgeries to u set $\mathsf{restricted}_u$
$\mathbf{ch}_u$	"done" when challenge from u	$-$ prevents an exposure of $\overline{u}$
	"forbidden" when z_u is exposed	$-$ prevents a challenge from u
$\mathcal{X}_u$	when $\overline{u}$ is exposed	$-$ prevents challenges until $r_u = \mathcal{X}_u$
$\mathsf{restricted}_u$	true when trivial forgery to u	$-$ prevents challenges from u
		$+$ (c, ad) from u not added to $\mathbf{ctable}_{\overline{u}}$
		$\pm$ show decryption of (c, ad) sent to u
		$+$ $\textsc{Exp}$ calls to u always allowed and will not change other variables

Fig. 8. Table summarizing some important variables in game AEAC. A "$-$" indicates a way in which the behavior of the adversary is being restricted. A "$+$" indicates a way in which the behavior of the adversary is being enabled.

ciphertext to u after exposing $\overline{u}$. Once u has received a different ciphertext than was sent by $\overline{u}$ there is no reason to think that u should be able to decrypt ciphertexts sent by $\overline{u}$ or send its own ciphertexts to $\overline{u}$. As such, if u is restricted (i.e. $\mathsf{restricted}_u = \mathsf{true}$) we will not add its ciphertexts to $\mathbf{ctable}_{\overline{u}}$, we will always show the true output when u attempts to decrypt ciphertexts given to it by the adversary (even if they were sent by $\overline{u}$), and if the adversary asks to expose u we will return all of its secret state without setting any of the other variables that would restrict the actions the adversary is allowed to take.

The above describes how $\mathsf{restricted}_u$ allows some attacks. Now we discuss how it prevents attacks like $\mathcal{D}_{3.1}$ and $\mathcal{D}_{3.2}$. Once the adversary has sent its own ciphertext to u we must assume that the adversary will be able to decrypt ciphertexts sent by u and able to send its own ciphertexts to u that will decrypt to non-$\perp$ values. The adversary could simply have "replaced" $\overline{u}$ with itself. To address this we prevent all challenge queries from u, and decryptions performed by u are always given back to the adversary regardless of the secret bit.

Informal description of the security game. In [27] we provide a thorough written description of our security model to facilitate high-level understanding of it. For intricate security definitions like ours there is often ambiguity or inconsistency in subtle corner cases of the definition when written out fully in text. As such this description should merely be considered an informal aid while the pseudocode of Fig. 7 is the actual definition.

Comparison to recent definitions. The three recents works we studied while deciding how to write our security definition were [14], [7], and [26]. Their settings were all distinct, but each presented security models that involve different "stages" of keys. All three works made distinct decisions in how to address challenges in different stages. In [27] we discuss these decisions, noting that they result in qualitatively identical but quantitatively distinct definitions.

6 Construction of a secure channel

6.1 Our construction

We are not aware of any secure channels that would meet (or could easily be modified to meet) our security notion. The "closest" (for some unspecified, informal notion of distance) is probably the Signal Double Ratchet Algorithm. However, it relies on symmetric authenticated encryption for both privacy and integrity so it is inherently incapable of achieving our strong notion of security. Later, we describe an attack against a variant of our proposed construction that uses symmetric primitives to exhibit the sorts of attacks that are unavoidable when using them. A straightforward variant of this attack would also apply against the Double Ratchet Algorithm.

In this section we construct our cryptographic channel and motivate our design decisions by giving attacks against variants of the channel. In Section 6.2 we will prove its security by reducing it to that of its underlying components.

The idea of our scheme is as follows. Both parties will keep track of a transcript of the messages they have sent and received, τ_s and τ_r. These will be included as a part of every ciphertext and verified before a ciphertext is accepted. On seeing a new ciphertext the appropriate transcript is updated to be the hash of the ciphertext (note that the old transcript is part of this ciphertext, so the transcript serves as a record of the entire conversation). Sending transcripts (vector of τ_s) are stored until the other party has acknowledged receiving a more recent transcript.

For authenticity, every time a user sends a ciphertext they authenticate it with a digital signature and include in it the verification key for the signing key that they will use to sign the next ciphertext they send. Any time a user receives a ciphertext they will use the new receiving transcript produced to update their current signing key.

For privacy, messages will be encrypted using public-key encryption. With every ciphertext the sender will include the encryption key for a new decryption key they have generated. Decryption keys are stored until the other party has acknowledged receiving a more recent encryption key. The encryption will use as a label all of the extra data that will be included with the ciphertext (i.e. a sending counter, a receiving counter, an associated data string, a new verification key, a new encryption key, a receiving transcript, and a sending transcript). The formal definition of our channel is as follows.

Cryptographic channel $\mathsf{SCH}[\mathsf{DS}, \mathsf{PKE}, \mathsf{H}]$. Let DS be a key-updatable digital signature scheme, PKE be a key-updating public-key encryption scheme, and H be

```
Algorithm SCh.Init
$(sk_{\mathcal{I}}, vk_{\mathcal{R}}) \leftarrow\!\$\ \mathsf{DS.Kg}$ ; $(ek_{\mathcal{I}}, \mathbf{dk}_{\mathcal{R}}[0]) \leftarrow\!\$\ \mathsf{PKE.Kg}$
$(sk_{\mathcal{R}}, vk_{\mathcal{I}}) \leftarrow\!\$\ \mathsf{DS.Kg}$ ; $(ek_{\mathcal{R}}, \mathbf{dk}_{\mathcal{I}}[0]) \leftarrow\!\$\ \mathsf{PKE.Kg}$
$hk \leftarrow\!\$\ \mathsf{H.Kg}$ ; $\tau_r \leftarrow \varepsilon$ ; $\boldsymbol{\tau}_s[0] \leftarrow \varepsilon$ ; $s \leftarrow r \leftarrow r^{ack} \leftarrow 0$
$st_{\mathcal{I}} \leftarrow (s, r, r^{ack}, sk_{\mathcal{I}}, vk_{\mathcal{I}}, ek_{\mathcal{I}}, \mathbf{dk}_{\mathcal{I}}, hk, \tau_r, \boldsymbol{\tau}_s)$
$st_{\mathcal{R}} \leftarrow (s, r, r^{ack}, sk_{\mathcal{R}}, vk_{\mathcal{R}}, ek_{\mathcal{R}}, \mathbf{dk}_{\mathcal{R}}, hk, \tau_r, \boldsymbol{\tau}_s)$
Return $(st_{\mathcal{I}}, st_{\mathcal{R}})$

Algorithm SCh.Send$(st, ad, m)$
$(s, r, r^{ack}, sk, vk, ek, \mathbf{dk}, hk, \tau_r, \boldsymbol{\tau}_s) \leftarrow st$ ; $s \leftarrow s + 1$
$(sk', vk') \leftarrow\!\$\ \mathsf{DS.Kg}$ ; $(ek', \mathbf{dk}[s]) \leftarrow\!\$\ \mathsf{PKE.Kg}$
$\ell \leftarrow (s, r, ad, vk', ek', \tau_r, \boldsymbol{\tau}_s[s-1])$
$(ek, c') \leftarrow\!\$\ \mathsf{PKE.Enc}(ek, \ell, m)$
$v \leftarrow (c', \ell)$ ; $\sigma \leftarrow\!\$\ \mathsf{DS.Sign}(sk, v)$
$c \leftarrow (\sigma, v)$ ; $\boldsymbol{\tau}_s[s] \leftarrow \mathsf{H.Ev}(hk, c)$
$st \leftarrow (s, r, r^{ack}, sk', vk, ek, \mathbf{dk}, hk, \tau_r, \boldsymbol{\tau}_s)$
Return $(st, c)$

Algorithm SCh.Recv$(st, ad, c)$
$(s, r, r^{ack}, sk, vk, ek, \mathbf{dk}, hk, \tau_r, \boldsymbol{\tau}_s) \leftarrow st$
$(\sigma, v) \leftarrow c$ ; $(c', \ell) \leftarrow v$
$(s', r', ad', vk', ek', \tau_r', \tau_s') \leftarrow \ell$
If $s' \neq r + 1$ or $\tau_r' \neq \boldsymbol{\tau}_s[r']$ or $\tau_s' \neq \tau_r$ or $ad \neq ad'$ then return $(st, \perp)$
$(vk'', t) \leftarrow \mathsf{DS.Vrfy}(vk, \sigma, v, \boldsymbol{\tau}_s[r^{ack}+1, \ldots, r'])$
If not $t$ then return $(st, \perp)$
$r \leftarrow r + 1$ ; $r^{ack} \leftarrow r'$ ; $(\mathbf{dk}[r'], m) \leftarrow\!\$\ \mathsf{PKE.Dec}(\mathbf{dk}[r'], \ell, c')$
$\boldsymbol{\tau}_s[0, \ldots, r'-1] \leftarrow \perp$ ; $\mathbf{dk}[0, \ldots, r'-1] \leftarrow \perp$
$\tau_r \leftarrow \mathsf{H.Ev}(hk, c)$ ; $sk \leftarrow\!\$\ \mathsf{DS.UpdSk}(sk, \tau_r)$
$st \leftarrow (s, r, r^{ack}, sk, vk', ek', \mathbf{dk}, hk, \tau_r, \boldsymbol{\tau}_s)$
Return $(st, m)$
```

Fig. 9. Construction of channel $\mathsf{SCh} = \mathsf{SCH}[\mathsf{DS}, \mathsf{PKE}, \mathsf{H}]$ from function family H, key-updatable digital signature scheme DS, and key-updating public-key encryption scheme PKE.

a family of functions. We build a cryptographic channel $\mathsf{SCh} = \mathsf{SCH}[\mathsf{DS}, \mathsf{PKE}, \mathsf{H}]$ as defined in Fig. 9.

A user's state st_u, among other values, contains counters s_u, r_u, r_u^{ack}. Here, s_u is the number of messages that u sent to $\overline{u}$, and r_u is the number of messages they received back from $\overline{u}$. The counter r_u^{ack} stores the last value of $r_{\overline{u}}$ in a ciphertext received by u (i.e. the index of the last ciphertext that u believes $\overline{u}$ has received and acknowledged). This counter is used to ensure that prior to running a signature verification algorithm, the vertification key vk is updated with respect to the same transcripts as the signing key sk (at the time it was used to produce the signature). Note that algorithm $\mathsf{DS.Vrfy}$ returns (vk'', t)

where t is the result of verifying that σ is a valid signature for v with respect to verification key vk'' (using the notation convention from Section 3).

Inefficiencies of SCh. A few aspects of SCh are less efficient than one would a priori hope. The state maintained by a user u (specifically the tables $\mathbf{dk}_u$ and $\boldsymbol{\tau}_{s,u}$) is not constant in size, but instead grows linearly with the number of ciphertexts that u sent to $\overline{u}$ without receiving a reply back. Additionally, when DS is instantiated with the particular choice of DS that we define in [27] the length of the ciphertext sent by a user u will grow linearly in the number of ciphertexts that u has received since the last time they sent a ciphertext. When PKE is instantiated with the scheme we define in [27] there is an extra state being stored that is linear in the number of ciphertexts that u has sent since it last received a ciphertext. Such inefficiencies would be unacceptable for a protocol like TLS or SSH, but in our motivating context of messaging is it plausible that they are acceptable. Each message is human generated and the state gets "refreshed" regularly if the two users regularly reply to one another. One could additionally consider designing an app to regularly send an empty message whose sole purpose is state refreshing. We leave as interesting future work improving on the efficiency of our construction.

Design decisions. We will now discuss attacks against different variants of SCh. This serves to motivate the decisions made in its design and give intuition for why it achieves the desired security. Several steps in the security proof of this construction can be understood by noting which of these attacks are ruled out in the process.

The attacks are shown in Fig. 10 and Fig. 11. The first several attacks serve to demonstrate that Ch.Send must use a sufficient amount of randomness (shown in $\mathcal{D}_a$, $\mathcal{D}_b$, $\mathcal{D}_c$) and that H needs to be collision resistant (shown in $\mathcal{D}_b$, $\mathcal{D}_c$). The next attack shows why our construction would be insecure if we did not use labels with PKE (shown in $\mathcal{D}_d$). Then we provide two attacks showing why the keys of DS and PKE need to be updated (shown in $\mathcal{D}_e$, $\mathcal{D}_f$). Then we show an attack that arises if multiple valid signatures can be found for the same string (shown in $\mathcal{D}_g$). Finally, we conclude with attacks that would apply if we used symmetric instead of asymmetric primitives to build SCh (shown in $\mathcal{D}_h$, $\mathcal{D}_i$).

Scheme with insufficient sending entropy. Any scheme whose sending algorithm has insufficient entropy will necessarily be insecure. For simplicity let SCh_1 be a variant of SCh such that SCh_1.Send is deterministic (the details of how we are making it deterministic do not matter). We can attack both the message privacy and the integrity of such a scheme.

Consider the adversary $\mathcal{D}_a$. It exposes $\mathcal{I}$, encrypts the message 1 locally, and then sends a challenge query to $\mathcal{I}$ asking for the encryption of either 1 or 0. By comparing the ciphertext it produced to the one returned by SEND it can determine which message was encrypted, learning the secret bit. We have

```
Adversary $\mathcal{D}_a^{\text{SEND},\text{RECV},\text{EXP}}$
$(st, z, \eta) \leftarrow \text{EXP}(\mathcal{I}, \varepsilon)$
$(st, c) \leftarrow \mathsf{SCh}_1.\mathsf{Send}(st, \varepsilon, 1)$
$c' \leftarrow \text{SEND}(\mathcal{I}, 0, 1, \varepsilon)$
If $c = c'$ then return 1
Return 0
```

```
Adversary $\mathcal{D}_b^{\text{SEND},\text{RECV},\text{EXP}}$
$(st, z, \eta) \leftarrow \text{EXP}(\mathcal{I}, \varepsilon)$
$(st, c) \leftarrow\$ \mathsf{Ch}.\mathsf{Send}(st, \varepsilon, 1)$
$m \leftarrow \text{RECV}(\mathcal{R}, c, \varepsilon)$
$c \leftarrow \text{SEND}(\mathcal{I}, 1, 1, \varepsilon)$
$c \leftarrow \text{SEND}(\mathcal{R}, 1, 1, \varepsilon)$
$m \leftarrow \text{RECV}(\mathcal{I}, c, \varepsilon)$
If $m = \perp$ then return 1
Return 0
```

```
Adversary $\mathcal{D}_c^{\text{SEND},\text{RECV},\text{EXP}}$
$(st, z, \eta) \leftarrow \text{EXP}(\mathcal{I}, \varepsilon)$
$(st, c) \leftarrow\$ \mathsf{Ch}.\mathsf{Send}(st, \varepsilon, 1)$
$m \leftarrow \text{RECV}(\mathcal{R}, c, \varepsilon)$
$c \leftarrow \text{SEND}(\mathcal{I}, 1, 1, \varepsilon)$
$(st, z, \eta) \leftarrow \text{EXP}(\mathcal{R}, \varepsilon)$
$(st, c) \leftarrow\$ \mathsf{Ch}.\mathsf{Send}(st, \varepsilon, 1)$
$m \leftarrow \text{RECV}(\mathcal{I}, c, \varepsilon)$
If $m = \perp$ then return 1 else return 0
```

```
Adversary $\mathcal{D}_d^{\text{SEND},\text{RECV},\text{EXP}}$
$(st, z, \eta) \leftarrow \text{EXP}(\mathcal{I}, \varepsilon)$
$c \leftarrow \text{SEND}(\mathcal{I}, 0, 1, \varepsilon)$
$(s, r, r^{ack}, sk, vk, ek, \mathbf{dk}, hk, \tau_r, \boldsymbol{\tau}_s) \leftarrow st$
$(\sigma, (c', (s, r, ad, vk', ek', \tau_r, \tau_s))) \leftarrow c$
$v \leftarrow (c', (s, r, 1^{128}, vk', ek', \tau_r, \tau_s))$
$\sigma \leftarrow\$ \mathsf{DS}.\mathsf{Sign}(sk, v)$
$m \leftarrow \text{RECV}(\mathcal{R}, (\sigma, v), 1^{128})$
If $m = 1$ then return 1 else return 0
```

Fig. 10. Attacks against variants of SCh.

$\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{SCh}_1}(\mathcal{D}_a) = 1$. This attack is fairly straightforward and will be ruled out by the security of PKE in our proof without having to be addressed directly.

The attacks against integrity are more subtle. They are explicitly addressed in the first game transition of our proof. Let $\mathsf{Ch} = \mathsf{SCh}_1$ and consider adversaries $\mathcal{D}_b$ and $\mathcal{D}_c$. They both start by doing the same sequence of operations: expose $\mathcal{I}$, use its secret state to encrypt and send message 1 to $\mathcal{R}$, then ask $\mathcal{I}$ to produce an encryption of 1 for $\mathcal{R}$ (which will be the same ciphertext as above, because $\mathsf{SCh}_1.\mathsf{Send}$ is deterministic). Now $\mathsf{restricted}_{\mathcal{R}} = \mathsf{true}$ because oracle RECV was called on a trivially fogeable ciphertext that was not produced by oralce SEND. But $\mathcal{R}$ has received the exact same ciphertext that $\mathcal{I}$ sent. Different attacks are possible from this point.

Adversary $\mathcal{D}_b$ just asks $\mathcal{R}$ to send a message and forwards it along to $\mathcal{I}$. Since $\mathcal{R}$ was restricted the ciphertext does not get added to $\mathbf{ctable}_{\mathcal{I}}$ so it can be used to discover the secret bit. We have $\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{SCh}_1}(\mathcal{D}_b) = 1$. Adversary $\mathcal{D}_c$ exposes $\mathcal{R}$ and uses the state it obtains to create its own forgery to $\mathcal{I}$. It then returns 1 or 0 depending on whether RECV returns the correct decryption or $\perp$. This attack succeeds because exposing $\mathcal{R}$ when it is restricted will not set any of the variables that would typically prevent the adversary from winning by creating a forgery. We have $\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{SCh}_1}(\mathcal{D}_c) = 1$. We have not shown it, but another message privacy attack at this point (instead of proceeding as $\mathcal{D}_b$ or $\mathcal{D}_c$) could have asked for another challenge query from $\mathcal{I}$, exposed $\mathcal{R}$, and used the exposed state to trivially determine which message was encrypted.

Scheme without collision-resistant hashing. If it is easy to find collisions in H then we can attack the channel by causing both parties to have matching transcripts despite having seen different sequences of ciphertexts. For concreteness let SCh_2 be a variant of our scheme using a hash function that outputs 0^{128} on all inputs. Let $\mathsf{Ch} = \mathsf{SCh}_2$ and again consider adversaries $\mathcal{D}_b$ and $\mathcal{D}_c$. We no longer expect the ciphertexts that they produce locally to match the ciphertexts returned by $\mathcal{I}$. However, they will have the same hash value and thus produce the same transcript $\tau_{r,\mathcal{R}} = 0^{128} = \tau_{s,\mathcal{I}}$. Consequently, $\mathcal{R}$ still updates its signing key in the same way regardless of whether it receives the ciphertext produced by $\mathcal{I}$ or the ciphertext locally generated by adversary. So the messages subsequently sent by $\mathcal{R}$ will still be accepted by $\mathcal{I}$. We have $\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{SCh}_2}(\mathcal{D}_b) = 1$ and $\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{SCh}_2}(\mathcal{D}_c) = 1$.

Scheme without PKE labels. Let SCh_3 be a variant of SCh that uses a public-key encryption scheme that does not accept labels and consider adversary $\mathcal{D}_d$. It exposes $\mathcal{I}$ and asks $\mathcal{I}$ for a challenge query. It then uses the state it exposed to trivially modify the ciphertext sent from $\mathcal{I}$ (we chose to have it change ad from ε to 1^{128}) and sends it to $\mathcal{R}$. Since the ciphertext sent to $\mathcal{R}$ has different associated data than the one sent by $\mathcal{I}$ the adversary will be given the decryption of this ciphertext. But without the use of labels this decryption by PKE is independent of the associated data and will thus reveal the true decryption of the challenge ciphertext to $\mathcal{I}$. We have $\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{SCh}_3}(\mathcal{D}_d) = 1$.

Schemes without key updating. We will now show why it is necessary to define new forms of PKE and DS for our construction.

Let SCh_4 be a variant of SCh that uses a digital signature scheme that does not update its keys. Consider adversary $\mathcal{D}_e$. It exposes $\mathcal{I}$, then queries Send for $\mathcal{I}$ to send a message to $\mathcal{R}$, but uses the exposed secrets to replace it with a locally produced ciphertext c. It calls Recv for $\mathcal{R}$ with c, which sets $\mathsf{restricted}_{\mathcal{R}} = \mathsf{true}$. Since the signing key is not updated in SCh_4, the adversary now exposes $\mathcal{R}$ to obtain a signing key whose signatures will be accepted by $\mathcal{I}$. It uses this to forge a ciphertext to $\mathcal{I}$ to learn the secret bit. We have $\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{SCh}_4}(\mathcal{D}_e) = 1$.

Let SCh_5 be a variant of SCh that uses a public-key encryption scheme that does not update its keys. Consider adversary $\mathcal{D}_f$. It exposes $\mathcal{I}$ and uses this to send $\mathcal{R}$ a different ciphertext than is sent by $\mathcal{I}$ (setting $\mathsf{restricted}_{\mathcal{R}} = \mathsf{true}$). Since the decryption key is not updated, the adversary now exposes $\mathcal{R}$ to obtain a decryption key that can be used to decrypt a challenge ciphertext sent by $\mathcal{I}$. We have $\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{SCh}_5}(\mathcal{D}_f) = 1$.

Scheme with non-unique signatures. Let SCh_6 be a variant of our scheme using a digital signature scheme that does not have unique signatures. For concreteness, assume that $\sigma \,\|\, sk$ is a valid signature whenever σ is. Then consider adversary $\mathcal{D}_g$. It exposes $\mathcal{I}$ and has $\mathcal{I}$ send a challenge ciphertext. Then it modifies the ciphertext by changing the signature and forwards this modified ciphertext on to $\mathcal{R}$. The adversary is given back the true decryption of this ciphertext (because it was changed) which trivially reveals the secret bit of the game (here it is impor-

```
Adversary $\mathcal{D}_e^{\text{Send,Recv,Exp}}$
$(st, z, \eta) \leftarrow \text{Exp}(\mathcal{I}, \varepsilon)$
$c_\mathcal{I} \leftarrow \text{Send}(\mathcal{I}, 1, 1, \varepsilon)$
$(\sigma, (c', (s, r, ad, vk_\mathcal{I}, ek_\mathcal{I}, \tau_r, \tau_s))) \leftarrow c_\mathcal{I}$
$(st, c) \leftarrow\$ \mathsf{SCh}_4.\mathsf{Send}(st, \varepsilon, 0)$
$m \leftarrow \text{Recv}(\mathcal{R}, c, \varepsilon)$
$(st, z, \eta) \leftarrow \text{Exp}(\mathcal{R}, \varepsilon)$
$(s, r, r^{ack}, sk, vk, ek, \mathbf{dk}, hk, \tau_r, \boldsymbol{\tau}_s) \leftarrow st$
$\tau_r \leftarrow \mathsf{H.Ev}(hk, c_\mathcal{I})$
$st \leftarrow (s, r, r^{ack}, sk, vk_\mathcal{I}, ek_\mathcal{I}, \mathbf{dk}, hk, \tau_r, \boldsymbol{\tau}_s)$
$(st, c) \leftarrow\$ \mathsf{SCh}_4.\mathsf{Send}(st, \varepsilon, 1)$
$m \leftarrow \text{Recv}(\mathcal{I}, c, \varepsilon)$
If $m = \bot$ then return 1 else return 0

Adversary $\mathcal{D}_f^{\text{Send,Recv,Exp}}$
$(st, z, \eta) \leftarrow \text{Exp}(\mathcal{I}, \varepsilon)$
$(\sigma, (c', \ell)) \leftarrow \text{Send}(\mathcal{I}, 0, 1, \varepsilon)$
$(st, c) \leftarrow\$ \mathsf{SCh}_5.\mathsf{Send}(st, \varepsilon, 0)$
$m \leftarrow \text{Recv}(\mathcal{R}, c, \varepsilon)$
$(st, z, \eta) \leftarrow \text{Exp}(\mathcal{R}, \varepsilon)$
$(s, r, r^{ack}, sk, vk, ek, \mathbf{dk}, hk, \tau_r, \boldsymbol{\tau}_s) \leftarrow st$
$(dk, m) \leftarrow\$ \mathsf{PKE.Dec}(\mathbf{dk}[0], \ell, c')$
If $m = 1$ then return 1 else return 0

Adversary $\mathcal{D}_g^{\text{Send,Recv,Exp}}$
$(st, z, \eta) \leftarrow \text{Exp}(\mathcal{I}, \varepsilon)$
$(\sigma, v) \leftarrow \text{Send}(\mathcal{I}, 0, 1, \varepsilon)$
$(s, r, r^{ack}, sk, vk, ek, \mathbf{dk}, hk, \tau_r, \boldsymbol{\tau}_s) \leftarrow st$
$m \leftarrow \text{Recv}(\mathcal{R}, (\sigma \parallel sk, v), \varepsilon)$
If $m = 1$ then return 1
Return 0

Adversary $\mathcal{D}_h^{\text{Send,Recv,Exp}}$
$(st, z, \eta) \leftarrow \text{Exp}(\mathcal{I}, \varepsilon)$
$(s, r, r^{ack}, sk, vk, ek, \mathbf{dk}, hk, \tau_r, \boldsymbol{\tau}_s) \leftarrow st$
$st \leftarrow (s, r, r^{ack}, vk, sk, ek, \mathbf{dk}, hk, \tau_r, \boldsymbol{\tau}_s)$
$(st, c) \leftarrow\$ \mathsf{SCh}_7.\mathsf{Send}(st, \varepsilon, 0)$
$m \leftarrow \text{Recv}(\mathcal{I}, c, \varepsilon)$
If $m = \bot$ then return 1
Return 0

Adversary $\mathcal{D}_i^{\text{Send,Recv,Exp}}$
$(st, z, \eta) \leftarrow \text{Exp}(\mathcal{I}, \varepsilon)$
$(\sigma, (c', \ell)) \leftarrow \text{Send}(\mathcal{I}, 0, 1, \varepsilon)$
$(s, r, r^{ack}, sk, vk, ek, \mathbf{dk}, hk, \tau_r, \boldsymbol{\tau}_s) \leftarrow st$
$m \leftarrow\$ \mathsf{PKE.Dec}(ek, \ell, c')$
If $m = 1$ then return 1
Return 0
```

Fig. 11. Attacks against variants of SCh.

tant that the signature is not part of the label used for encryption/decryption). We have $\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{SCh}_6}(\mathcal{D}_g) = 1$.

Scheme with symmetric primitives. Let SCh_7 be a variant of our scheme that uses a MAC instead of a digital signature scheme (e.g. $vk = sk$ always, and vk is presumably no longer sent in the clear with the ciphertext). Consider adversary $\mathcal{D}_h$. It simply exposes $\mathcal{I}$ and then uses $\mathcal{I}$'s vk to send a message to $\mathcal{I}$. This trivially allows it to determine the secret bit. Here we used that PKE will decrypt any ciphertext to a non-$\bot$ value. We have $\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{SCh}_7}(\mathcal{D}_h) = 1$.

Similarly let SCh_8 be a variant of our scheme that uses symmetric encryption instead of public-key encryption (e.g. $ek = dk$ always, and ek is presumably no longer sent in the clear with the ciphertext). Adversary $\mathcal{D}_i$ exposes user $\mathcal{I}$ and then uses the corresponding ek to decrypt a challenge message encrypted by $\mathcal{I}$. We have $\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{SCh}_8}(\mathcal{D}_i) = 1$.

Stated broadly, a scheme that relies on symmetric primitives will not be secure because a user will know sufficient information to send a ciphertext that they would themselves accept or to read a message that they sent to the other user. Our security notion requires that this is not possible.

6.2 Security theorem

The following theorem bounds the advantage of an adversary breaking the AEAC security of SCh using the advantages of adversaries against the CR security of H, the UFEXP and UNIQ security of DS, the INDEXP security of PKE, and the min-entropy of DS and PKE.

Theorem 1. *Let* DS *be a key-updatable digital signature scheme,* PKE *be a key-updating public-key encryption scheme, and* H *be a family of functions. Let* $\mathsf{SCh} = \mathsf{SCH}[\mathsf{DS}, \mathsf{PKE}, \mathsf{H}]$*. Let* $\mathcal{D}$ *be an adversary making at most* q_{SEND} *queries to its* SEND *oracle,* q_{RECV} *queries to its* RECV *oracle, and* q_{EXP} *queries to its* EXP *oracle. Then we can build adversaries* $\mathcal{A}_{\mathsf{H}}$*,* $\mathcal{A}_{\mathsf{DS}}$*,* $\mathcal{B}_{\mathsf{DS}}$*, and* $\mathcal{A}_{\mathsf{PKE}}$ *such that*

$$\mathsf{Adv}^{\mathsf{aeac}}_{\mathsf{SCh}}(\mathcal{D}) \leq 2 \cdot (q_{\text{SEND}} \cdot 2^{-\mu} + \mathsf{Adv}^{\mathsf{cr}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{H}}) + \mathsf{Adv}^{\mathsf{ufexp}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{DS}}) + \\ + \mathsf{Adv}^{\mathsf{uniq}}_{\mathsf{DS}}(\mathcal{B}_{\mathsf{DS}})) + \mathsf{Adv}^{\mathsf{indexp}}_{\mathsf{PKE}}(\mathcal{A}_{\mathsf{PKE}})$$

where $\mu = \mathrm{H}_\infty(\mathsf{DS.Kg}) + \mathrm{H}_\infty(\mathsf{PKE.Kg}) + \mathrm{H}_\infty(\mathsf{PKE.Enc})$*. Adversary* $\mathcal{A}_{\mathsf{DS}}$ *makes at most* $q_{\text{SEND}} + 2$ *queries to its* NEWUSER *oracle,* q_{SEND} *queries to its* SIGN *oracle, and* q_{EXP} *queries to its* EXP *oracle. Adversary* $\mathcal{B}_{\mathsf{DS}}$ *makes at most* $q_{\text{SEND}} + 2$ *queries to its* NEWUSER *oracle. Adversary* $\mathcal{A}_{\mathsf{PKE}}$ *makes at most* $q_{\text{SEND}} + 2$ *queries to its* NEWUSER *oracle,* q_{SEND} *queries to its* ENC *oracle,* q_{RECV} *queries to its* DEC *oracle,* $q_{\text{SEND}} + 2$ *queries to its* EXPDK *oracle, and* $\min\{q_{\text{EXP}}, q_{\text{SEND}} + 1\}$ *queries to its* EXPRAND *oracle. Adversaries* $\mathcal{A}_{\mathsf{H}}$*,* $\mathcal{A}_{\mathsf{DS}}$*,* $\mathcal{B}_{\mathsf{DS}}$*, and* $\mathcal{A}_{\mathsf{PKE}}$ *all have runtime about that of* $\mathcal{D}$*.*

The proof is in [27]. It broadly consists of two stages. The first stage of the proof (consisting of three game transitions) argues that the adversary will not be able to forge a ciphertext to an unrestricted user except by exposing the other user. This argument is justified by a reduction to an adversary $\mathcal{A}_{\mathsf{DS}}$ against the security of the digital signature scheme. However, care must be taken in this reduction to ensure that $\mathcal{D}$ cannot induce behavior in $\mathcal{A}_{\mathsf{DS}}$ that would result in $\mathcal{A}_{\mathsf{DS}}$ cheating in the digital signature game. Addressing this possibility involves arguing that $\mathcal{D}$ cannot predict any output of SEND (from whence the min-entropy term in the bound arises) and that it cannot find any collisions in the hash function H.

Once this stage is complete the output of RECV no longer depends on the secret bit b, so we move to using the security of PKE to argue that $\mathcal{D}$ cannot use SEND to learn the value of the secret bit. This is the second stage of the proof. But prior to this reduction we have to make one last argument using the security of DS. Specifically we show that, given a ciphertext (σ, v), the adversary will not be able to find a new signature σ' such that (σ', v) will be accepted by the receiver (otherwise since $\sigma \neq \sigma'$, oracle RECV would return the true decryption of this ciphertext which would be the same as the decryption of the original ciphertext and thus allow a trivial attack). Having done this, the reduction to the security of PKE is straightforward.

Acknowledgments

We thank Mihir Bellare for extensive discussion on preliminary versions of this paper. We thank the CRYPTO 2018 reviewers for their comments. Jaeger and Stepanovs were supported in part by NSF grants CNS-1717640 and CNS-1526801.

References

1. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. *EUROCRYPT 2000*.
2. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. *FOCS 1997*.
3. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. *CRYPTO 1998*.
4. M. Bellare, T. Kohno, and C. Namprempre. Breaking and provably repairing the ssh authenticated encryption scheme: A case study of the encode-then-encrypt-and-mac paradigm. *ACM Transactions on Information and System Security (TISSEC)*, 7(2):206–241, 2004.
5. M. Bellare and S. K. Miner. A forward-secure digital signature scheme. *CRYPTO 1999*.
6. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. *EUROCRYPT 2006*.
7. M. Bellare, A. C. Singh, J. Jaeger, M. Nyayapati, and I. Stepanovs. Ratcheted encryption and key exchange: The security of messaging. *CRYPTO 2017*.
8. M. Bellare and B. Yee. Forward-security in private-key cryptography. *CT-RSA 2003*.
9. A. Boldyreva, J. P. Degabriele, K. G. Paterson, and M. Stam. Security of symmetric encryption in the presence of ciphertext fragmentation. *EUROCRYPT 2012*.
10. N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use pgp. In *ACM Workshop on Privacy in the Electronic Society, 2004*.
11. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. *FOCS 2001*.
12. R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. *Journal of Cryptology*, 20(3):265–294, July 2007.
13. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. *EUROCRYPT 2001*.
14. K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the Signal messaging protocol. *Proc. IEEE European Symposium on Security and Privacy (EuroS&P), 2017*.
15. K. Cohn-Gordon, C. Cremers, and L. Garratt. On post-compromise security. In *IEEE Computer Security Foundations Symposium (CSF), 2016*.
16. Y. Desmedt and Y. Frankel. Threshold cryptosystems. *CRYPTO 1989*.
17. W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, June 1992.
18. Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-insulated public key cryptosystems. *EUROCRYPT 2002*.

19. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong key-insulated signature schemes. *PKC 2003*.
20. Y. Dodis, W. Luo, S. Xu, and M. Yung. Key-insulated symmetric key cryptography and mitigating attacks against cryptographic cloud software. *ASIACCS 2012*.
21. T. P. (editor) and M. Marlinspike. The double ratchet algorithm. `https://whispersystems.org/docs/specifications/doubleratchet/`, Nov. 20, 2016.
22. M. Fischlin, F. Günther, G. A. Marson, and K. G. Paterson. Data is a stream: Security of stream-based channels. *CRYPTO 2015*.
23. C. Gentry and A. Silverberg. Hierarchical ID-based cryptography. *ASIACRYPT 2002*.
24. M. D. Green and I. Miers. Forward secure asynchronous messaging from puncturable encryption. In *IEEE Symposium on Security and Privacy, 2015*.
25. C. G. Günther. An identity-based key-exchange protocol. *EUROCRYPT 1989*.
26. F. Günther and S. Mazaheri. A formal treatment of multi-key channels. *CRYPTO 2017*.
27. J. Jaeger and I. Stepanovs. Optimal Channel Security Against Fine-Grained State Compromise: The Safety of Messaging. Cryptology ePrint Archive, Report 2018/XYZ, 2018. To appear.
28. H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). *CRYPTO 2001*.
29. A. Langley. Pond. GitHub repository, README.md, `https://github.com/agl/pond/commit/7bb06244b9aa121d367a6d556867992d1481f0c8`, 2012.
30. G. A. Marson and B. Poettering. Security notions for bidirectional channels. *IACR Trans. Symm. Cryptol.*, 2017(1):405–426, 2017.
31. M. Mignotte. How to share a secret? *EUROCRYPT 1982*.
32. C. Namprempre. Secure channels based on authenticated encryption schemes: A simple characterization. *ASIACRYPT 2002*.
33. Open Whisper Systems. Signal protocol library for java/android. GitHub repository, `https://github.com/WhisperSystems/libsignal-protocol-java`, 2017.
34. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). *ACM PODC 1991*.
35. B. Poettering and P. Rösler. Ratcheted key exchange, revisited. Cryptology ePrint Archive, Report 2018/296, 2018. `https://eprint.iacr.org/2018/296`.
36. P. Rogaway. Authenticated-encryption with associated-data. *ACM CCS 2002*.
37. P. Rogaway and T. Shrimpton. A provable-security treatment of the key-wrap problem. *EUROCRYPT 2006*.
38. A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, Nov. 1979.
39. V. Shoup. On formal models for secure key exchange. Cryptology ePrint Archive, Report 1999/012, 1999. `http://eprint.iacr.org/1999/012`.
40. V. Shoup. A proposal for an iso standard for public key encryption. Cryptology ePrint Archive, Report 2001/112, 2001. `https://eprint.iacr.org/2001/112`.
41. M. Tompa and H. Woll. How to share a secret with cheaters. *Journal of Cryptology*, 1(2):133–138, 1988.
42. N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. SoK: Secure messaging. *IEEE Symposium on Security and Privacy, 2015*.
43. WhatsApp Blog. Connecting one billion users every day. `https://blog.whatsapp.com/10000631/Connecting-One-Billion-Users-Every-Day`, July 26, 2017.