Symmetric Signcryption and E2EE Group Messaging in Keybase

Joseph Jaeger[1](✉), Akshaya Kumar[1], and Igors Stepanovs[2]

[1] School of Cybersecurity and Privacy, Georgia Institute of Technology, Atlanta, GA, USA
{josephjaeger,akshayakumar}@gatech.edu
[2] Riga, Latvia
igors.stepanovs@gmail.com
https://cc.gatech.edu/~josephjaeger/, https://cc.gatech.edu/~akumar805/, https://igors.org/

Abstract. We introduce a new cryptographic primitive called symmetric signcryption, which differs from traditional signcryption because the sender and recipient share a secret key. We prove that a natural composition of symmetric encryption and signatures achieves strong notions of security against attackers that can learn and control many keys. We then identify that the core encryption algorithm of the Keybase encrypted messaging protocol can be modeled as a symmetric signcryption scheme. We prove the security of this algorithm, though our proof requires assuming non-standard, brittle security properties of the underlying primitives.

1 Introduction

Keybase is a suite of encryption tools. It encompasses a public-key directory, an instant messenger, and a cloud storage service. Keybase was launched in 2014. In February 2020, it reported having accumulated more than 1.1M user accounts [27]. In May 2020, Keybase was acquired by Zoom. At the time, Zoom issued a public statement [36] saying that the Keybase's team was meant to play a critical part in building scalable end-to-end encryption for Zoom. The acquisition appears to have put an end to an active development of new Keybase features, but as of February 2024 it keeps receiving regular maintenance updates.

Instant Messaging in Keybase. Keybase implements its own end-to-end encrypted instant messaging protocol. This protocol is designed to support large groups. One-on-one chats are treated as group chats and hence use the same protocol. The protocol also allows to send large files as encrypted attachments in chat. It is impossible to opt out of end-to-end encryption in Keybase. In this work we analyze the security of this protocol.

The Keybase client is open source [24], but the server is not. Our security analysis primarily relies on the source code. Keybase also provides the "Keybase Book" website [22] with excellent documentation that explains its cryptographic design. The only prior security analysis of Keybase was done by NCC Group in

© International Association for Cryptologic Research 2024
M. Joye and G. Leander (Eds.): EUROCRYPT 2024, LNCS 14653, pp. 283–312, 2024.
https://doi.org/10.1007/978-3-031-58734-4_10

2019 [31], which broadly looked at the security of the entire Keybase ecosystem. In comparison, we provide an in-depth analysis of a single component in Keybase.

Encrypted Group Chats. In this work we consider a setting in which an arbitrary number of users can form a group. All group members share a key for a symmetric encryption scheme. Each instant message within the group is encrypted with this key. Let us use g to denote the identity of a group and K_g to denote the key shared between the members of this group. In Keybase, every member of group g uses the same long-term key K_g to encrypt their outgoing chat messages. Each message is encrypted only once, simultaneously for all recipients. The resulting ciphertext is then broadcast to all members of the group.

The *Sender Keys* protocol [7,28] can be seen as building on this basic design idea. In *Sender Keys*, every member of the group owns a distinct symmetric encryption key; they share it with other group members. Each outgoing message is encrypted with the sender's own key, and the resulting ciphertext is broadcast to the group. Furthermore, each key is used to encrypt only a single message, and immediately afterwards a new key is derived to be used for the next encryption. So every group member tracks every other member's current encryption key, decrypting each incoming ciphertext with the corresponding sender's key and subsequently replacing it with an appropriately derived new key. Variants of the *Sender Keys* protocol are used in the *Signal* [28], *WhatsApp* [34], and *Matrix* [1,2] messengers. In addition, the *Messaging Layer Security* (MLS) [8] protocol contains a component called *FS-GAEAD* [3] or *TreeDEM* [33] that similarly uses a sender's key to encrypt and broadcast a message (but its overall design significantly differs from design of the *Sender Keys* protocol).

An encrypted group chat protocol should provide at least confidentiality and integrity of communication, with respect to an attacker that is not a member of the group. In part, this could be achieved by building the protocol from a symmetric encryption scheme that satisfies some notion of authenticated-encryption security. But care is needed to also prevent undesired message replays, reordering, or drops. These requirements are specific to a stateful protocol and do not necessarily follow from properties provided by the underlying stateless scheme.

Sender Authentication in Group Chats. Consider a group chat protocol that is built from a single symmetric encryption scheme and where every symmetric key is known to all group members. In such a protocol, group members are able to impersonate each other. This is true regardless of whether each group uses a single shared encryption key or has each member own a distinct encryption key. To prevent group members from impersonating each other, it is natural to use a digital signature scheme. Let us use u to denote the identity of a user and sk_u to denote this user's signing key for a digital signature scheme.

What is a sound way to compose a symmetric encryption scheme with a digital signature scheme? Let us consider two sequential compositions of a signing algorithm Sign with an encryption algorithm $\mathsf{Encrypt}$. We call the resulting schemes $\mathsf{Sign\text{-}then\text{-}Encrypt}$ and $\mathsf{Encrypt\text{-}then\text{-}Sign}$, and we show them in Fig. 1. The $\mathsf{Sign\text{-}then\text{-}Encrypt}$ scheme first signs a message m to obtain its digital signature s and then encrypts (s, m) to obtain and return a symmetric ciphertext

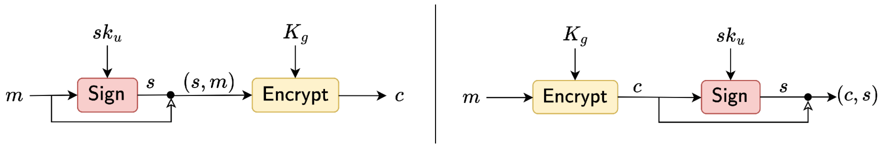


Fig. 1. Warmup schemes obtained by composing digital signatures with symmetric encryption. **Left pane:** Sign-then-Encrypt. **Right pane:** Encrypt-then-Sign.

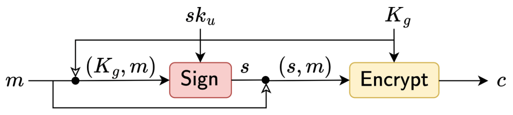


Fig. 2. A high-level representation of the SealPacket scheme in Keybase.

c. The Encrypt-then-Sign scheme first encrypts m as c and then computes a signature s over c; it returns the pair (c, s).

These compositions closely mirror those that are commonly used to build signcryption [5], which is a standard cryptographic primitive that combines digital signatures with public-key encryption [35], except we replace public-key encryption with symmetric encryption. It is well known that the corresponding compositions for signcryption are not secure in the multi-user setting, unless some effort is taken to bind together the message with the sender and recipient identities [5]. The standard advice is to always sign the recipient's identity and always encrypt the sender's identity. Our basic schemes in Fig. 1 would intuitively suffer from similar issues and benefit from similar countermeasures. However, the exact details would depend on what kind of security one expects from these schemes, so we defer this discussion.

The *Sender Keys* protocol [7] prescribes to sign a symmetric ciphertext; and this is indeed done by the *Signal*, *WhatsApp*, and *Matrix* messengers. The MLS protocol [8] protocol prescribes to encrypt a digital signature with a symmetric encryption scheme. So either protocol can be seen as using some variant of Encrypt-then-Sign or Sign-then-Encrypt as a subroutine. We will now discuss that Keybase can be seen as extending both of these basic schemes.

SealPacket: Sign-then-Encrypt in Keybase. Keybase uses a variant of the basic Sign-then-Encrypt scheme. It signs the symmetric encryption key along with the plaintext, meaning it signs (K_g, m) instead of just m. The resulting scheme is called SealPacket and is shown in Fig. 2. In the source code, the decision to sign K_g is explained as follows [26]:

simply using encryption and signing together isn't good enough ... the inner layer needs to assert something about the outer layer ... a better approach is to mix the outer key into the inner crypto, so that it's impossible to forget to check it ... That means the inner signing layer needs to assert

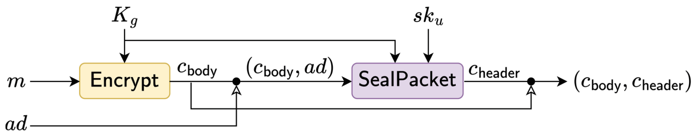


Fig. 3. A high-level representation of the BoxMessage scheme in Keybase.

the encryption key ... We don't need to worry about whether the signature might leak the encryption key either, because the signature gets encrypted.

Keybase uses SealPacket to encrypt the following three types of plaintexts: (1) a metadata header that is automatically created and sent along with every chat message, (2) a file that is sent as an attachment in chat, and (3) an arbitrary string chosen by a chat bot (for secure server-side storage of bot data).

BoxMessage: Encrypt-then-Sign in Keybase. The BoxMessage scheme in Keybase is a variant of the basic Encrypt-then-Sign scheme. This scheme, unlike SealPacket, is used only for one purpose: to encrypt the body of a chat message. So we denote by c_{body} the symmetric ciphertext that is created in the inner (encryption) layer of BoxMessage. The BoxMessage scheme extends the basic Encrypt-then-Sign scheme in two ways and is shown in Fig. 3. First, it takes an associated-data field ad and signs (c_{body}, ad) instead of just c_{body}. Second, rather than use a signature scheme, BoxMessage uses SealPacket to sign (c_{body}, ad).

Keybase uses the auxiliary-data field ad to authenticate a metadata header for the chat message. This header contains the group's identity and sender's identity among multiple other values. The data in ad is sent in plain over the network (along with c_{body}), meaning that SealPacket is not meant to provide confidentiality of ad. Indeed, the Keybase documentation explains that SealPacket is used to provide the confidentiality of the *signature* over (c_{body}, ad) [23]:

fields in the header aren't secret from the server, and it actually needs to know several of them ... The reason for sign-then-encrypting/signencrypting the header is instead to keep the signature itself private. Even though the server knows who's talking to whom, because it's delivering all the messages, it's better that it can't prove what it knows.

Interestingly, BoxMessage reuses the group's symmetric encryption key K_g between its calls to Encrypt and SealPacket. As mentioned above, SealPacket will itself first sign K_g and then run another instance of Encrypt with K_g as the key. In total, the same value of K_g is therefore used in 3 distinct contexts.

Symmetric Signcryption. We define *symmetric signcryption* as a new cryptographic primitive that combines symmetric encryption with digital signatures. We capture the setting where every user owns a signing key pair and in each group all users share a single symmetric encryption key. The encryption key is long-term, meaning it can be used an arbitrary number of times, simultaneously by all members of the group. This will allow us to formalize and analyze the SealPacket and BoxMessage schemes.

We note that the use of sender-specific encryption keys in the *Sender Keys* [7] and MLS [8] protocols can also be captured by symmetric signcryption. Indeed, in either protocol all symmetric encryption keys can seen as being independently sampled, and each individual key is used only once. This can be thought of as a collection of "one-time-use" symmetric signcryption schemes.

We adapt the standard syntax of (asymmetric) signcryption to suit our setting, defining algorithms SigEnc and VerDec. They both take explicit sender and group identities, nonces, and associated data.

Security of Symmetric Signcryption. We define two security notions for symmetric signcryption. The *out-group authenticated encryption* (OAE) security requires confidentiality and integrity of communication against an adversary that does not know the symmetric key of the group it attacks. This is required to hold even against an adversary that can assign to every user an arbitrary (possibly malformed) digital signature key pair. The *in-group unforgeability* (IUF) security requires unforgeability of messages sent by users whose signing keys are not known to the adversary. This is required to hold even against an adversary that can assign to every group an arbitrary (possibly malformed) symmetric key.

We show how to extend the basic Sign-then-Encrypt scheme, by carefully incorporating user and group identifiers, to achieve both of our security notions. We assume strong unforgeability of the underlying digital signature scheme and authenticated-encryption security of the underlying encryption scheme.

Implementation of SealPacket and BoxMessage. In the source code of the Keybase client, SealPacket is implemented in [26] and BoxMessage is implemented in [25]. These schemes are instantiated with the nonce-based authenticated encryption scheme XSalsa20-Poly1305 [15,16] and the digital signature scheme Ed25519 [17,18]. They also use SHA-512 and SHA-256 which does not significantly affect the design of either scheme so we omit discussing it here, but in the main body of the paper, we attempt to formalize both schemes precisely.

Keybase implements four versions of the BoxMessage scheme: V1, V2, V3, and V4. V1 is deprecated; the Keybase client allows to receive but not send messages that use V1. V2 is the default version that we formalize and analyze in this work. V3 is the same as V2, except it supports exploding messages; the body of an exploding message is encrypted using an ephemeral key instead of K_g. V4 is the same as V3, except it makes all group members use a dummy (zero) signing key and instead authenticate messages using pairwise MACs.

Provable Security Analysis. We model BoxMessage and SealPacket as symmetric signcryption schemes and provide formal reductions for their IUF and OAE security. Our analysis is done in a concrete security framework [11], and in a multi-key setting; we state precise bounds on the advantage of an attacker. The analysis of BoxMessage largely encompasses that of SealPacket, because BoxMessage uses SealPacket in a modular way. So we focus on the analysis of BoxMessage here. The main challenges arise from using K_g in 3 distinct contexts.

First, we aim to show it is hard to switch the context of the XSalsa20-Poly1305 ciphertexts c_{body} and c_{header}. Both are encrypted using the same key K_g, so there is a risk that an attacker could forge a valid encryption of some body-plaintext m from a known encryption of some header-plaintext $(s, c_{\mathsf{body}}, ad)$, or vice versa. To rule out such attacks, we rely on an observation that every application-layer message m that is queried to be encrypted by BoxMessage is encoded in a specific way, whereas every header-plaintext $(s, c_{\mathsf{body}}, ad)$ is expected to start with a valid Ed25519 signature. Based on the specification of Ed25519 we show (in the ROM and GGM) that it is hard to cast the encoding used in m as a valid Ed25519 signature (with respect to any verification key of adversary's choice).

Second, we need to show that XSalsa20-Poly1305 provides authenticated encryption even for certain messages derived from its secret key. This arises because in SealPacket the XSalsa20-Poly1305 key K_g is first signed with Ed25519 and then the resulting signature is encrypted using XSalsa20-Poly1305 under the same key. Here again we rely on the specification of Ed25519. An Ed25519 signature depends on two SHA-512 hash values of the message that is being signed, but it does not depend on the signed message beyond that. We use this (in the ROM) to eliminate the need to consider key-dependent messages and hence only require XSalsa20-Poly1305 to provide the standard notion of authenticated encryption.

We do not know any way to avoid the above analysis. The necessity to use non-standard security notions appears to be inherently implied by the design decisions made in Keybase. This could have been avoided (e.g. with out Sign-then-Encrypt scheme). Overall, our reductions (in the ROM and GGM) rely on the AEAD security of XSalsa20-Poly1305, collision resistance of SHA-256 and SHA-512, and strong unforgeability of Ed25519. We note that Keybase uses the version of Ed25519 that was recently shown to be SUF-CMA secure [10,20].

Limitations of Our Work. Our analysis of Keybase is intentionally narrow in scope. We perform an in-depth, algorithmic analysis of specific chat components that can be modeled as symmetric signcryption. Other analysis is outside the scope of our work, such as whether these algorithms are secure against timing attacks and whether they provide protection against message replays, reordering, or drops when used within the broader stateful chat protocol. More broadly, our analysis does not explicitly cover many other applications of cryptography in Keybase, including other versions of BoxMessage, encryption of attachments or bot data, the initial key exchange used to agree on group keys, the public-key directory used to share user keys, and the cloud storage service. These applications are important for the overall security of Keybase, and have the potential to interplay with each other in subtle ways. For example, user signing keys are used for multiple tasks in Keybase. We believe appropriate context separation is used for these purposes (e.g. all messages signed in SealPacket start with "Keybase-Chat-2"). If not, subtle cross-application attacks may be possible.

Related Work. The Hybrid Public-Key Encryption (HPKE) standard is specified in RFC 9180 [9]. Alwen, Janneck, Kiltz, and Lipp [4] analyze the "pre-shared key" modes from RFC 9180. They cast the $\mathsf{HPKE}_{\mathsf{AuthPSK}}$ mode as an asymmetric signcryption scheme that is augmented with a pre-shared symmetric key, and they define the corresponding security notions. They analyze the security that is achieved by $\mathsf{HPKE}_{\mathsf{AuthPSK}}$ depending on which combinations of keys are secure. Our definitions are similar in the sense that both works define a signcryption-type primitive that in addition uses a symmetric key. However, the algorithms in [4] use one more set of keys, and the definitions in [4] are stated in the two-user setting. In essence, our primitives are similar in form, but are tailored to be used as tools in different settings.

2 Preliminaries

We use standard pseudocode notation and assume familiarity with hash functions, random oracles, nonce-based encryption, and digital signatures. Collision resistance is defined by $\mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{CR}}) = \Pr[\mathsf{H}(x) = \mathsf{H}(y) \wedge x \neq y \,:\, (x, y) \leftarrow\!\!\$\; \mathcal{A}_{\mathsf{CR}}]$.

2.1 Standard Security Notions in a Multi-key Setting

Key Management Oracles. Throughout this work we consider multi-key security notions. Adversaries in security games will be provided with three types of key management oracles. These oracles will allow (1) sampling new honest (i.e. challenge) keys, (2) exposing existing honest keys, and (3) adding corrupt keys of the adversary's choice. When an honest key is exposed it becomes corrupt, but it was initially sampled from a correct key distribution. In contrast, when an adversary adds its own corrupt key, such a key could be maliciously crafted in an arbitrary way. In basic security notions an adversary cannot benefit from crafting corrupt keys, because no challenge queries are permitted with respect to such keys. This changes for more complex systems built from more than one keyed primitive when some security is required to hold even if some underlying secrets are exposed. The ability to use malicious keys was modeled in prior work on (asymmetric) signcryption [14], and will be needed in this work.

Our security model for symmetric signcryption in Sect. 3 will define two sets of key management oracles. The set of *user oracles* $\mathrm{U} = \{\textsc{NewHonUser}, \textsc{ExposeUser}, \textsc{NewCorrUser}\}$ will manage the keys for a digital signature scheme, whereas the set of *group oracles* $\mathrm{G} = \{\textsc{NewHonGroup}, \textsc{ExposeGroup}, \textsc{NewCorrGroup}\}$ will manage the keys for a nonce-based encryption scheme. We adopt the same terminology and notation across all of the multi-key security notions; each notion for an *asymmetric* primitive will define a set of user oracles U, and each notion for a *symmetric* primitive will define a set of group oracles G. For consistency, we include oracles for adding corrupt keys even when an adversary cannot benefit from using them. When simulating user oracles in a security reduction, we write $\textsc{SimU}$ to denote the set $\{\textsc{SimNewHonUser}, \textsc{SimExposeUser}, \textsc{SimNewCorrUser}\}$ and do similarly for group oracles.

Nonce-Based Authenticated Encryption. Consider game $\mathcal{G}^{\mathsf{AEAD}}$ of Fig. 4 for nonce-based encryption scheme NE and adversary $\mathcal{A}_{\mathsf{AEAD}}$. The advantage of

$\mathcal{A}_{\mathsf{AEAD}}$ in breaking the AEAD security of NE is defined as $\mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}}) = 2 \cdot \Pr[\mathcal{G}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}})] - 1$. The game samples a challenge bit b, and $\mathcal{A}_{\mathsf{AEAD}}$ is required to guess it. Adversary $\mathcal{A}_{\mathsf{AEAD}}$ is given the group oracles G, encryption oracle Enc, and decryption oracle Dec. Among the group oracles, NewHonGroup creates new groups with honestly generated NE keys, ExposeGroup reveals the keys of existing groups, and NewCorrGroup instantiates new corrupt groups with NE keys of $\mathcal{A}_{\mathsf{AEAD}}$'s choice. We require NE to be nonce-misuse resistant [30], meaning that no challenge message m is allowed to be queried across two distinct calls to Enc with respect to the same set of g, n, ad. A corrupt group key can only be used to call Enc with $m_0 = m_1$, and a group key cannot be exposed after it has been used in Enc with $m_0 \neq m_1$. The decryption oracle Dec takes g, n, c, ad as input and decrypts this to the corresponding plaintext m. Following the all-in-one style of [30,32], it returns $\perp$ if $b = 0$, and it returns m otherwise. This oracle never decrypts a ciphertext with an exposed group's key, and it never decrypts ciphertexts previously produced by Enc (with the same g, c, ad).

```
Game $\mathcal{G}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}})$
$b \leftarrow\$ \{0,1\}$ ; $b' \leftarrow\$ \mathcal{A}^{\mathrm{G,Enc,Dec}}_{\mathsf{AEAD}}$ ; Return $b = b'$

Enc$(g, n, m_0, m_1, ad)$
require $\mathsf{K}[g] \neq \perp$ and $|m_0| = |m_1|$
require $\forall d \in \{0,1\}, (g, n, m_d, ad) \notin N_d$
If $m_0 \neq m_1$ then
    If $\mathsf{group\_is\_corrupt}[g]$ then return $\perp$
    $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_b, ad)$
$N_0 \leftarrow N_0 \cup \{(g, n, m_0, ad)\}$
$N_1 \leftarrow N_1 \cup \{(g, n, m_1, ad)\}$
$C \leftarrow C \cup \{(g, n, c, ad)\}$ ; Return $c$

Dec$(g, n, c, ad)$
require $\mathsf{K}[g] \neq \perp$ and $\neg\mathsf{group\_is\_corrupt}[g]$
require $(g, n, c, ad) \notin C$
$m \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c, ad)$
If $b = 0$ then return $\perp$ else return $m$
```

```
Game $\mathcal{G}^{\mathsf{KR}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{KR}})$
$\mathcal{A}^{\mathrm{G,Enc,Dec,Guess}}_{\mathsf{KR}}$ ; Return win

Enc$(g, n, m, ad)$
require $\mathsf{K}[g] \neq \perp$
require $(g, n, m, ad) \notin N$
$c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m, ad)$
$N \leftarrow N \cup \{(g, n, m, ad)\}$ ; Return $c$

Dec$(g, n, c, ad)$
require $\mathsf{K}[g] \neq \perp$
$m \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c, ad)$
Return $m$

Guess$(K)$
If $\exists g$: $(\mathsf{K}[g] = K$ and
$\neg\mathsf{group\_is\_corrupt}[g])$ then
    $\mathsf{win} \leftarrow \mathsf{true}$
```

```
NewHonGroup$(g)$
require $\mathsf{K}[g] = \perp$
$\mathsf{K}[g] \leftarrow\$ \{0,1\}^{\mathsf{NE.kl}}$

ExposeGroup$(g)$
require $\mathsf{K}[g] \neq \perp$ $\boxed{\text{and } \neg\mathsf{chal}[g]}$
$\mathsf{group\_is\_corrupt}[g] \leftarrow \mathsf{true}$
Return $\mathsf{K}[g]$

NewCorrGroup$(g, K)$
require $\mathsf{K}[g] = \perp$
$\mathsf{group\_is\_corrupt}[g] \leftarrow \mathsf{true}$
$\mathsf{K}[g] \leftarrow K$
```

Fig. 4. Left pane: Game defining authenticated-encryption security of a nonce-based encryption scheme NE. **Right pane:** Game defining key-recovery security of NE. **Bottom pane:** Group oracles G = {NewHonGroup, ExposeGroup, NewCorrGroup} that are provided to an adversary in either game, except that the $\boxed{\text{boxed}}$ code only appears in the AEAD security game.

Key-Recovery Security of NE. Consider game $\mathcal{G}^{\mathsf{KR}}$ of Fig. 4 for nonce-based encryption scheme NE and adversary $\mathcal{A}_{\mathsf{KR}}$. The advantage of $\mathcal{A}_{\mathsf{KR}}$ in breaking the KR security of NE is defined as $\mathsf{Adv}^{\mathsf{KR}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{KR}}) = \Pr[\mathcal{G}^{\mathsf{KR}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{KR}})]$.

```
Game $\mathcal{G}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})$
  $(u, m, s) \leftarrow\$ \mathcal{A}^{\mathrm{U},\textsc{Sign}}_{\mathsf{SUFCMA}}$ ; If $\mathsf{vk}[u] = \bot$ then return $\mathsf{false}$
  $\mathsf{win}_1 \leftarrow \neg\mathsf{user\_is\_corrupt}[u]$ ; $\mathsf{win}_2 \leftarrow ((u, m, s) \notin S)$
  $\mathsf{win}_3 \leftarrow \mathsf{DS.Ver}(\mathsf{vk}[u], m, s)$
  Return $\mathsf{win}_1$ and $\mathsf{win}_2$ and $\mathsf{win}_3$

$\textsc{Sign}(u, m)$
  require $\mathsf{sk}[u] \neq \bot$
  $s \leftarrow\$ \mathsf{DS.Sig}(\mathsf{sk}[u], m)$
  $S \leftarrow S \cup \{(u, m, s)\}$
  Return $s$

$\textsc{NewHonUser}(u)$
  require $\mathsf{sk}[u] = \mathsf{vk}[u] = \bot$
  $(\mathsf{sk}[u], \mathsf{vk}[u]) \leftarrow\$ \mathsf{DS.Kg}$
  Return $\mathsf{vk}[u]$

$\textsc{ExposeUser}(u)$
  require $\mathsf{sk}[u] \neq \bot$
  $\mathsf{user\_is\_corrupt}[u] \leftarrow \mathsf{true}$
  Return $\mathsf{sk}[u]$

$\textsc{NewCorrUser}(u, sk, vk)$
  require $\mathsf{sk}[u] = \mathsf{vk}[u] = \bot$
  $\mathsf{user\_is\_corrupt}[u] \leftarrow \mathsf{true}$
  $\mathsf{sk}[u] \leftarrow sk$ ; $\mathsf{vk}[u] \leftarrow vk$
```

Fig. 5. Game defining strong unforgeability of a digital signature scheme DS, where $\mathrm{U} = \{\textsc{NewHonUser}, \textsc{ExposeUser}, \textsc{NewCorrUser}\}$.

Strong Unforgeability of Digital Signatures. Consider game $\mathcal{G}^{\mathsf{SUFCMA}}$ of Fig. 5 for signature scheme DS and adversary $\mathcal{A}_{\mathsf{SUFCMA}}$. The advantage of $\mathcal{A}_{\mathsf{SUFCMA}}$ in breaking the SUFCMA security of DS is defined as $\mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}}) = \Pr[\mathcal{G}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})]$.

3 Symmetric Signcryption

In this section we define syntax and security for multi-user symmetric signcryption. In symmetric signcryption, a user encrypts messages using their signing key and a symmetric key shared by a group of users. We want that nobody outside a group can learn what messages are being encrypted, and nobody at all can forge a message as having come from someone other than themself.

Syntax. A symmetric signcryption scheme SS specifies algorithms $\mathsf{SS.UserKg}$, $\mathsf{SS.SigEnc}$, $\mathsf{SS.VerDec}$, where $\mathsf{SS.VerDec}$ is deterministic. These algorithm use syntax $(sk, vk) \leftarrow\$ \mathsf{SS.UserKg}$, $c \leftarrow\$ \mathsf{SS.SigEnc}(g, K_g, u, sk_u, n, m, ad)$, and $m \leftarrow \mathsf{SS.VerDec}(g, K_g, u, vk_u, n, c, ad)$. Associated to SS is a group-key length $\mathsf{SS.gkl} \in \mathbb{N}$, a nonce space $\mathsf{SS.NS}$, a plaintext space $\mathsf{SS.MS} \subseteq \{0,1\}^*$, and an associated-data space $\mathsf{SS.AD}$. The user's key generation algorithm $\mathsf{SS.UserKg}$ returns a key pair (sk, vk) where sk is a signing key and vk is the corresponding verification key. The signcryption algorithm $\mathsf{SS.SigEnc}$ takes a group's identifier $g \in \{0,1\}^*$ and its symmetric key $K_g \in \{0,1\}^{\mathsf{SS.gkl}}$, a sender's identifier $u \in \{0,1\}^*$ and its signing key sk_u, a nonce $n \in \mathsf{SS.NS}$, a plaintext $m \in \mathsf{SS.MS}$, and associated data $ad \in \mathsf{SS.AD}$; it returns a signcryption ciphertext c. The deterministic unsigncryption algorithm $\mathsf{SS.VerDec}$ takes $g, K_g, u, vk_u, n, c, ad$, where vk_u is the verification key of the sender u; it returns a plaintext $m \in \{0,1\}^* \cup \{\perp\}$, where $\perp$ indicates a failure to recover a plaintext. We say that SS is *deterministic* $\mathsf{SS.SigEnc}$ is deterministic. Correctness is defined in the natural way.

3.1 In-Group Unforgeability

The strongest variant of in-group unforgeability requires that an attacker cannot modify anything about ciphertexts. We also capture weaker variants. For example, the $\mathsf{SealPacket}$ encryption algorithm in Keybase (as defined in Sect. 4) uses a signing key to bind its ciphertexts to a group's symmetric key but not to a group's identifier. So we parameterize our security definition in order to capture the type of authenticity that is as restrictive as possible except for allowing (what can be described as) cross-group forgeries.

IUF Game. Consider game $\mathcal{G}^{\mathsf{IUF}}$ of Fig. 6, defined for symmetric signcryption scheme SS, ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}$, and adversary $\mathcal{A}_{\mathsf{IUF}}$. The advantage of $\mathcal{A}_{\mathsf{IUF}}$ in breaking the IUF security of SS is defined as $\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{SS},\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) = \Pr[\mathcal{G}^{\mathsf{IUF}}_{\mathsf{SS},\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}})]$. Adversary $\mathcal{A}_{\mathsf{IUF}}$ is given access to user oracles U, group oracles G, encryption oracle SigEnc, and decryption oracle VerDec. Its goal is to set the win flag by forging a ciphertext for an honest user.

Among user oracles, NewHonUser creates honest users with honestly generated signing keys, NewCorrUser creates corrupt users with malicious signing keys, and ExposeUser exposes signing keys of existing users. Among group oracles, NewHonGroup creates honest groups with honestly sampled symmetric keys, NewCorrGroup creates corrupt groups with malicious symmetric keys, and ExposeGroup exposes symmetric keys of existing groups. Oracles NewHonGroup and NewCorrGroup take as input a set users identifying the new group's users; the encryption and decryption oracles then disallow queries that match a group to a non-member user. The user and group oracles use tables $\mathsf{user\_is\_corrupt}$ and $\mathsf{group\_is\_corrupt}$ in order to keep track of the users and groups whose keys are not secure, respectively. The IUF game never checks $\mathsf{group\_is\_corrupt}$, deliberately giving the adversary full control over group keys.

The encryption oracle SigEnc takes (g, u, n, m, ad) and returns a ciphertext c that is produced by running $\mathsf{SS.SigEnc}(g, \mathsf{K}[g], u, \mathsf{sk}[u], n, m, ad)$. Here note that the group and user keys $\mathsf{K}[g]$ and $\mathsf{sk}[u]$ are the only two inputs to $\mathsf{SS.SigEnc}$ that are not directly chosen by the adversary at the moment of querying the SigEnc oracle. At the end of each SigEnc query, the set C is updated to add the tuple $((g, u, n, m, ad), c)$ that can be interpreted as containing the input-output transcript of this query.

The decryption oracle VerDec takes (g, u, n, c, ad) and returns the message m that is recovered by running $\mathsf{SS.VerDec}(g, \mathsf{K}[g], u, \mathsf{vk}[u], n, c, ad)$. Keys $\mathsf{K}[g], \mathsf{sk}[g]$ are the only inputs to $\mathsf{SS.VerDec}$ not directly chosen by the adversary. If $m \neq \perp$, then the oracle determines if the current oracle query is a valid forgery and sets the win flag if so. In particular, VerDec builds the tuple $z = ((g, u, n, m, ad), c)$ with all input and output values of the current decryption query. It checks z against the set C that contains the input-output behavior

```
Game $\mathcal{G}^{\mathsf{IUF}}_{\mathsf{SS},\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}})$
$\mathcal{A}^{\mathrm{U},\mathrm{G},\textsc{SigEnc},\textsc{VerDec}}_{\mathsf{IUF}}$
Return $\mathsf{win}$

$\textsc{SigEnc}(g, u, n, m, ad)$
require $\mathsf{K}[g] \neq \bot$
require $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$c \leftarrow_{\$} \mathsf{SS.SigEnc}(g, \mathsf{K}[g], u, \mathsf{sk}[u], n, m, ad)$
$C \leftarrow C \cup \{((g, u, n, m, ad), c)\}$
Return $c$

$\textsc{VerDec}(g, u, n, c, ad)$
require $\mathsf{K}[g] \neq \bot$
require $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$m \leftarrow \mathsf{SS.VerDec}(g, \mathsf{K}[g], u, \mathsf{vk}[u], n, c, ad)$
If $m = \bot$ then return $\bot$
$z \leftarrow ((g, u, n, m, ad), c)$
If $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}(z, C)$ then return $m$
If $\neg\mathsf{user\_is\_corrupt}[u]$ then $\mathsf{win} \leftarrow \mathsf{true}$
Return $m$
```

```
Game $\mathcal{G}^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{OAE}})$
$b \leftarrow_{\$} \{0, 1\}$ ; $b' \leftarrow_{\$} \mathcal{A}^{\mathrm{U},\mathrm{G},\textsc{SigEnc},\textsc{VerDec}}_{\mathsf{OAE}}$
Return $b = b'$

$\textsc{SigEnc}(g, u, n, m_0, m_1, ad)$
require $\mathsf{K}[g] \neq \bot$ and $|m_0| = |m_1|$
require $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
require $\forall d \in \{0, 1\}, (g, u, n, m_d, ad) \notin N_d$
If $m_0 \neq m_1$ then
    If $\mathsf{group\_is\_corrupt}[g]$ then return $\bot$
    $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$c \leftarrow_{\$} \mathsf{SS.SigEnc}(g, \mathsf{K}[g], u, \mathsf{sk}[u], n, m_b, ad)$
$N_0 \leftarrow N_0 \cup \{(g, u, n, m_0, ad)\}$
$N_1 \leftarrow N_1 \cup \{(g, u, n, m_1, ad)\}$
$C \leftarrow C \cup \{((g, u, n, m_b, ad), c)\}$
$Q \leftarrow Q \cup \{((g, u, n, m_0, m_1, ad), c)\}$
Return $c$

$\textsc{VerDec}(g, u, n, c, ad)$
require $\mathsf{K}[g] \neq \bot$ and $\neg\mathsf{group\_is\_corrupt}[g]$
require $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$m \leftarrow \mathsf{SS.VerDec}(g, \mathsf{K}[g], u, \mathsf{vk}[u], n, c, ad)$
If $m = \bot$ then return $\bot$
$z \leftarrow ((g, u, n, m, ad), c)$
If $\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}}(z, C)$ then return $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}(z, Q)$
If $b = 0$ then return $\bot$ else return $m$
```

```
$\textsc{NewHonUser}(u)$
require $\mathsf{sk}[u] = \mathsf{vk}[u] = \bot$
$(\mathsf{sk}[u], \mathsf{vk}[u]) \leftarrow_{\$} \mathsf{SS.UserKg}$
Return $\mathsf{vk}[u]$

$\textsc{ExposeUser}(u)$
require $\mathsf{sk}[u] \neq \bot$
$\mathsf{user\_is\_corrupt}[u] \leftarrow \mathsf{true}$
Return $\mathsf{sk}[u]$

$\textsc{NewCorrUser}(u, sk, vk)$
require $\mathsf{sk}[u] = \mathsf{vk}[u] = \bot$
$\mathsf{user\_is\_corrupt}[u] \leftarrow \mathsf{true}$
$\mathsf{sk}[u] \leftarrow sk$ ; $\mathsf{vk}[u] \leftarrow vk$
```

```
$\textsc{NewHonGroup}(g, \mathsf{users})$
require $\mathsf{K}[g] = \bot$
$\mathsf{K}[g] \leftarrow_{\$} \{0, 1\}^{\mathsf{SS.gkl}}$
$\mathsf{members}[g] \leftarrow \mathsf{users}$

$\textsc{ExposeGroup}(g)$
require $\mathsf{K}[g] \neq \bot$ $\boxed{\text{and } \neg\mathsf{chal}[g]}$
$\mathsf{group\_is\_corrupt}[g] \leftarrow \mathsf{true}$
Return $\mathsf{K}[g]$

$\textsc{NewCorrGroup}(g, K, \mathsf{users})$
require $\mathsf{K}[g] = \bot$
$\mathsf{group\_is\_corrupt}[g] \leftarrow \mathsf{true}$
$\mathsf{K}[g] \leftarrow K$ ; $\mathsf{members}[g] \leftarrow \mathsf{users}$
```

Fig. 6. Left pane: Game defining in-group unforgeability IUF of a symmetric signcryption scheme SS with respect to a ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}$. **Right pane:** Game defining out-group authenticated-encryption security OAE of SS with respect to a ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}}$ and an output-guarding function $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$. **Bottom pane:** User oracles $\mathrm{U} = \{\textsc{NewHonUser}, \textsc{ExposeUser}, \textsc{NewCorrUser}\}$ and group oracles $\mathrm{G} = \{\textsc{NewHonGroup}, \textsc{ExposeGroup}, \textsc{NewCorrGroup}\}$ that are provided to an adversary in either game, except that the $\boxed{\text{boxed}}$ code only appears in the OAE security game.

```
$\underline{\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}(z, C)}$
Return $z \in C$

$\underline{\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}(z, C)}$
$((g, u, n, m, ad), c) \leftarrow z$
Return $\exists c'\colon ((g, u, n, m, ad), c') \in C$
```

```
$\underline{\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}(z, C)}$
$((g, u, n, m, ad), c) \leftarrow z$
Return $\exists g'\colon ((g', u, n, m, ad), c) \in C$

$\underline{\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}(z, C)}$
$((g, u, n, m, ad), c) \leftarrow z$
Return $\exists u'\colon ((g, u', n, m, ad), c) \in C$
```

Fig. 7. Sample ciphertext-triviality predicates which capture rules for deciding if a successfully decrypted VerDec query was trivially obtainable or forgeable.

of all the prior encryption queries. If z is determined to be trivially obtainable from the information in C, then VerDec exits early (with m as its output value); otherwise it sets the win flag. This check is performed by the *ciphertext-triviality predicate* $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}$. We will describe the syntax and the sample variants of $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}$ below.

Ciphertext-Triviality Predicates. The IUF security game is parameterized by ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}$ (we will also parameterize the OAE game with $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}$). Predicate $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}$ takes a tuple $z = ((g, u, n, m, ad), c)$ and a set C as input, where C contains tuples of the same format. Here z describes the input-output values of the current query to VerDec oracle and each element of C contains an input-output transcript of a prior SigEnc oracle query. Predicate $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}$ returns true if z is considered to be trivially forgeable based on the information in C and false otherwise.

In Fig. 7 we define several ciphertext-triviality predicates. Predicate $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ checks if $z \in C$, capturing the strongest possible level of authenticity. This requires that only prior outputs of SigEnc can be successfully queried to the VerDec oracle; any other successful decryption query causes the adversary to win the IUF game. This predicate can be thought of as making the IUF game capture the "strong" unforgeability of ciphertexts in our group setting. One could capture existential unforgeability by considering the predicate $\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}$ that does not allow the adversary to win by merely producing new ciphertexts that decrypt to some tuple (g, u, n, m, ad) previously queried to SigEnc. Predicates $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}$ and $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ capture the authenticity of schemes where a ciphertext encrypting (g, u, n, m, ad) is not bound to the group's identifier or to the user's identifier, respectively. We use $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$, $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}$ and $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ in our security analysis of Keybase. In this work, we do not use $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ with the IUF game – we need it for OAE.

3.2 Out-Group Authenticated Encryption

The strongest version of the out-group AE security requires that an attacker outside a chat group can neither learn any information about the exchanged messages, nor modify the exchanged ciphertexts in any way. We also capture weaker variants of this security notion. For example, the $\mathsf{SealPacket}$ encryption algorithm (as defined in Sect. 4) does not use a group's symmetric key to explicitly bind its ciphertexts to a user's signing key or a user's identifier when used in isolation. So we capture a variant of out-group AE security that is as restrictive as possible except for allowing an attacker to violate the sender's authenticity within any particular group.

OAE Game. Consider game $\mathcal{G}^{\mathsf{OAE}}$ of Fig. 6 for symmetric signcryption scheme SS, ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}$, output-guarding function $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$, and adversary $\mathcal{A}_{\mathsf{OAE}}$. The advantage in breaking the OAE security of SS is defined as $\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{OAE}}) = \Pr[\mathcal{G}^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{OAE}})]$. Adversary $\mathcal{A}_{\mathsf{OAE}}$ is given access to user and group oracles U and G and to the encryption and decryption oracles SigEnc and VerDec. The goal of the adversary is to guess the challenge bit b. Our security game is defined in the all-in-one style of [29,32], where an adversary can learn the challenge bit by forging a ciphertext to its decryption oracle.

The user and group oracles in the OAE game are defined as in the IUF game, except it does not allow calling the ExposeGroup oracle to expose the key of a group that was previously used for a left-or-right challenge-encryption query (as explained below). The OAE game never checks the contents of $\mathsf{user\_is\_corrupt}$, deliberately giving the adversary full control over user keys.

The encryption oracle SigEnc takes (g, u, n, m_0, m_1, ad) and returns a ciphertext c by running $\mathsf{SS.SigEnc}(g, \mathsf{K}[g], u, \mathsf{sk}[u], n, m_b, ad)$. The group and user keys $\mathsf{K}[g]$ and $\mathsf{sk}[u]$ are the only inputs to $\mathsf{SS.SigEnc}$ not directly chosen by the adversary querying the SigEnc oracle (and the encrypted message m_b depends on the challenge bit). The SigEnc query requires that $|m_0| = |m_1|$ and will only use insecure group keys for non-challenge encryptions (i.e. for $m_0 = m_1$). This SigEnc oracle captures nonce-misuse resistance [30], using the sets N_d to prevent trivial wins. At the end of SigEnc queries, the set C is updated to add the tuple $((g, u, n, m_b, ad), c)$, and the set Q is updated to add the tuple $((g, u, n, m_0, m_1, ad), c)$. Here the Q set can contain the input-output "transcript" of SigEnc queries from the adversary's point of view, whereas the set C is more informative because it contains the message that was actually encrypted. We will explain the purpose of these sets below.

The decryption oracle VerDec takes (g, u, n, c, ad) and returns the message m output by $\mathsf{SS.VerDec}(g, \mathsf{K}[g], u, \mathsf{vk}[u], n, c, ad)$. Keys $\mathsf{K}[g], \mathsf{sk}[g]$ are the only inputs to $\mathsf{SS.VerDec}$ not directly chosen by the adversary querying the VerDec oracle. The VerDec oracle disallows queries with corrupt group keys; if an adversary knows a group's key then it can decrypt ciphertexts for the group on its own. If $\mathsf{SS.VerDec}$ recovers a non-$\bot$ message m and the end of the VerDec oracle is reached, then the challenge bit is meant to be revealed through returning m if $b = 1$ and $\bot$ otherwise. However, this intuition is not precise; it depends on how VerDec responds to queries that are identified as being trivially forgeable. Similarly to how trivial forgeries were handled in the IUF game, here VerDec builds $z = ((g, u, n, m, ad), c)$ and uses a ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}}$ to check z against the set C from SigEnc. If z is considered *not* trivially obtainable from the information in C, then VerDec proceeds to its last instruction that returns $\perp$ or m depending on the challenge bit. Otherwise, VERDEC should return an output that does not depend on the challenge bit to prevent trivial wins. Such an output is produced by the *output-guarding function* $\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}$, i.e. VERDEC returns the output of $\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}(z, Q)$. We now describe the syntax and variants of $\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}$.

Output-Guarding Functions. The OAE game can be parameterized by different choices of an output-guarding function $\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}$. We define $\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}$ to take a tuple $z = ((g, u, n, m, ad), c)$ and a set Q as input, where Q contains tuples with the format $((g, u, n, m_0, m_1, ad), c)$. Here z describes the input-output values of a single query to the VERDEC oracle, and each element of C specifies the input-output of a prior SIGENC oracle query. At a high level, z contains the message m that was recovered during an ongoing VERDEC call, and m is the only value in z, Q not necessarily known by the adversary. One might want to define VERDEC to return m whenever the input is identified as a trivial forgery, but m could potentially trivially reveal the challenge bit. So one could roughly think of $\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}$ as the function that should enable VERDEC to return m when possible. However, it should determine – from z and Q – if m would trivially help the adversary win and then "guard" VERDEC against returning this m.

```
$\mathsf{func}_{\mathsf{out}}^{\perp}(z, Q)$
Return $\perp$

$\mathsf{func}_{\mathsf{out}}^{\mathsf{silence\text{-}with\text{-}m_1}}[\mathsf{pred}_{\mathsf{trivial}}](z, Q)$
For each $((g, u, n, m_0, m_1, ad), c) \in Q$ do
    If $m_0 \neq m_1$ then
        If $\mathsf{pred}_{\mathsf{trivial}}(z, \{((g, u, n, m_0, ad), c)\})$ then return $m_1$
        If $\mathsf{pred}_{\mathsf{trivial}}(z, \{((g, u, n, m_1, ad), c)\})$ then return $m_1$
$((g, u, n, m, ad), c) \leftarrow z$ ; Return $m$
```

Fig. 8. Sample output-guarding functions $\mathsf{func}_{\mathsf{out}}^{\perp}$ and $\mathsf{func}_{\mathsf{out}}^{\mathsf{silence\text{-}with\text{-}m_1}}$. Function $\mathsf{func}_{\mathsf{out}}^{\mathsf{silence\text{-}with\text{-}m_1}}$ is parameterized by a ciphertext-triviality predicate $\mathsf{pred}_{\mathsf{trivial}}$.

In Fig. 8 we define two output-guarding functions. The function $\mathsf{func}_{\mathsf{out}}^{\perp}$ always returns $\perp$. This provides no useful information to the adversary and so captures a comparatively weaker security notion. The function $\mathsf{func}_{\mathsf{out}}^{\mathsf{silence\text{-}with\text{-}m_1}}[\mathsf{pred}_{\mathsf{trivial}}]$ is parameterized by an arbitrary ciphertext-triviality predicate $\mathsf{pred}_{\mathsf{trivial}}$ and captures the following logic. For every element in Q that describes a challenge encryption (i.e. $m_0 \neq m_1$) performed by SIGENC, this function checks whether z is trivially forgeable based on the information that the adversary could have learned from the corresponding response. This is checked if z would be trivially forgeable for *both* choices of $b \in \{0, 1\}$ or only for *only one* choice of b. The output-guarding function returns m_1 when this condition passes. If no element of Q triggered the above, then the output-guarding function returns the m contained in z, i.e. the actual message recovered in VERDEC.

The Use of $\mathsf{func}_{\mathsf{out}}^{\mathsf{silence\text{-}with\text{-}m_1}}$ in Our Work. We target $\mathsf{func}_{\mathsf{out}}^{\mathsf{silence\text{-}with\text{-}m_1}}[\mathsf{pred}_{\mathsf{trivial}}]$ as the output-guarding function that provides the strongest possible security guarantees for the schemes that we analyze in this work. For every $\mathsf{pred}_{\mathsf{trivial}}$ we use,

```
$\underline{\mathsf{BoxMessage.SigEnc}(g, K_g, u, sk_u, n, m_{\mathsf{body}}, ad)}$  // where $n = (n_{\mathsf{body}}, n_{\mathsf{header}})$
$c_{\mathsf{body}} \leftarrow \mathsf{XSalsa20\text{-}Poly1305.Enc}(K_g, n_{\mathsf{body}}, m_{\mathsf{body}})$
$h_{\mathsf{body}} \leftarrow \mathsf{SHA\text{-}256}(n_{\mathsf{body}} \,\|\, c_{\mathsf{body}})$
$m_{\mathsf{header}} \leftarrow \langle ad, u, g, h_{\mathsf{body}} \rangle$
$c_{\mathsf{header}} \leftarrow \mathsf{SealPacket.SigEnc}(g, K_g, u, sk_u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon)$
Return $(c_{\mathsf{body}}, c_{\mathsf{header}})$

$\underline{\mathsf{SealPacket.SigEnc}(g, K_g, u, sk_u, n, m, ad)}$  // where $ad = \varepsilon$
$h \leftarrow \mathsf{SHA\text{-}512}(m)$ ; $m_s \leftarrow$ “Keybase-Chat-2” $\|\, \langle K_g, n, h \rangle$
$s \leftarrow \mathsf{Ed25519.Sig}(sk_u, m_s)$ ; $m_e \leftarrow s \,\|\, m$
$c \leftarrow \mathsf{XSalsa20\text{-}Poly1305.Enc}(K_g, n, m_e)$
Return $c$
```

Fig. 9. The BoxMessage and SealPacket algorithms used in Keybase for encrypting chat messages from a user to a group. Here g is the group's identifier, K_g is the symmetric key shared by all group members, u is the sender's identifier, and sk_u is the sender's signing key.

$\mathsf{pred}_{\mathsf{trivial}}(z, \{((g, u, n, m^*, ad), c)\})$ can only be true when z contains m^*. So for elements of Q with $m_0 \neq m_1$ only one of the two `if` conditions can pass, meaning it is necessary to silence the output. Otherwise the adversary can trivially win the game by building z, Q and evaluating $\mathsf{pred}_{\mathsf{trivial}}$ to distinguish between $b = 0$ or $b = 1$. (This attack assumes the adversary can always compute $\mathsf{pred}_{\mathsf{trivial}}(z, C)$ for SS, in spite of not knowing the challenge bit b that is needed to explicitly build C. This is true in all of our proofs.)

3.3 Symmetric Signcryption from Encryption and Signatures

In the full version, we introduce a provably secure version of Sign-then-Encrypt (StE). Its signcryption algorithm signs $s \leftarrow\!\!\$\; \mathsf{DS.Sig}(sk_u, \langle g, n, m, ad \rangle)$ then outputs ciphertext $c \leftarrow \mathsf{NE.Enc}(K_g, n, s \,\|\, m, \langle u, ad \rangle)$. We prove bounds of the form $\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{StE},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})$ and $\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{StE},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}},\mathsf{func}^{\perp}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{OAE}}) \leq \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}})$.

4 Keybase Chat Encryption as Symmetric Signcryption

We analyze the security of the cryptographic algorithm BoxMessage that Keybase uses to encrypt and authenticate chat messages from a sender to a group. BoxMessage combines multiple cryptographic primitives to offer end-to-end encrypted messaging. In particular it uses XSalsa20-Poly1305, SHA-256, SHA-512, and Ed25519 as building blocks. Within BoxMessage, the SealPacket subroutine encrypts and authenticates message headers. We show the pseudocode for these algorithms in Fig. 9. We omit the decryption algorithms

BoxMessage.VerDec and SealPacket.VerDec from Fig. 9 as Keybase's implementation of these algorithms follows naturally from the corresponding SigEnc algorithms. We define the VerDec algorithms explicitly in our formalazation of BoxMessage and SealPacket.

To formalize the security of BoxMessage, it is crucial to first identify the formal primitive underlying this algorithm and the security goals it aims to achieve. None of the existing primitives in literature seem to aptly model this object, but it is naturally captured by the symmetric signcryption primitive that we defined in Sect. 3. Similarly, SealPacket can also be modeled as a symmetric signcryption scheme from which BoxMessage is built. In this section, we present modular constructions that cast BoxMessage and SealPacket as symmetric signcryption schemes. We first provide a general overview of the two algorithms.

The BoxMessage Chat-Encryption Algorithm. The BoxMessage.SigEnc algorithm accepts the following inputs – group's identifier g, symmetric group key K_g, sender identifier u, sender signing key sk_u, nonce $n = (n_{\mathsf{body}}, n_{\mathsf{header}})$, message m_{body}, and associated data ad. It performs the following steps. First it calls XSalsa20-Poly1305.Enc to encrypt m_{body} using key K_g and nonce n_{body}, and obtains the ciphertext c_{body}. It builds header plaintext m_{header} as $\langle ad, u, g, h_{\mathsf{body}} \rangle$ (a unique encoding of ad, u, g, and hash $h_{\mathsf{body}} = \mathsf{SHA\text{-}256}(n_{\mathsf{body}} \,\|\, c_{\mathsf{body}})$). It then invokes SealPacket.SigEnc to encrypt m_{header} using sk_u, K_g, and n_{header}, and obtains the ciphertext c_{header}. Finally, it returns $(c_{\mathsf{body}}, c_{\mathsf{header}})$. To decrypt ciphertext $(c_{\mathsf{body}}, c_{\mathsf{header}})$, the algorithm BoxMessage.VerDec (not shown) ensures that c_{header} decrypts into the header plaintext m_{header} that is equal to the unique string $\langle ad, u, g, h_{\mathsf{body}} \rangle$ composed from the inputs of BoxMessage.VerDec. In Keybase, the sender identifier u is their username and the group identifier g is constructed canonically from the usernames of the group members.

The SealPacket Header-Encryption Algorithm. The SealPacket algorithm accepts the same inputs as BoxMessage, except it does not take associated data ad as input. We capture this by setting SealPacket.AD $= \{\varepsilon\}$, meaning $ad = \varepsilon$ is always true When SealPacket.SigEnc is called from BoxMessage.SigEnc, it encrypts chat headers. To encrypt m with nonce n, it starts by hashing m to obtain $h = \mathsf{SHA\text{-}512}(m)$. Then it builds an input m_s to the Ed25519 signature scheme by concatenating the prefix string "Keybase-Chat-2" with the unique encoding $\langle K_g, n, h \rangle$ of K_g, n, and h. It invokes Ed25519.Sig to produce a signature s over m_s using the signing key sk_u. Finally it calls XSalsa20-Poly1305.Enc to encrypt $m_e = s \,\|\, m$ using the key K_g and nonce n, and obtains the ciphertext c which is returned To decryption ciphertext c, the SealPacket.VerDec algorithm (not shown) first recovers m_e from c and then parses m_e to obtain $s \,\|\, m$. Note that m_e can be unambiguously parsed into $s \,\|\, m$ because Ed25519 produces fixed-length signatures. Then SealPacket.VerDec reconstructs m_s and ensures that s verifies as a valid signature for m_s under the sender's public key vk_u. We study the security of SealPacket in the context of the BoxMessage algorithm, but this is not the only context in which Keybase uses SealPacket. It is also used independently for the encryption of long strings and attachments. In the full version we detail other uses of SealPacket in Keybase.

Analysis Challenges. The descriptions of BoxMessage and SealPacket that we have given so far already present the following challenges in their analysis.

Key Reuse in BoxMessage. The same symmetric key K_g is used in BoxMessage and SealPacket. This violates the principle of key separation, which says that one should always use distinct keys for distinct algorithms and modes of operation. Without context separation, this potentially allows an attacker to forward ciphertexts produced by one algorithm to another. There is no explicit context separation, so our analysis will "extract" separation by making assumptions of Ed25519 and using low-level details of how messages are encoded.

Cyclic Key Dependency in SealPacket. The message m_s signed in SealPacket is derived from the symmetric group key K_g which is also used to encrypt the signature. This produces what is known as an "encryption cycle", a generalization of encrypting one's own key [19]. Standard AEAD security does not guarantee security when messages being encrypted depend on the key used for encryption. We use an extension of AEAD security allowing key-dependent messages and prove (in the random oracle model) that XSalsa20-Poly1305 achieves it for the particular key-dependent messages required.

Lack of Group/User Binding in SealPacket. By looking at the SealPacket algorithm in Fig. 9 we can see that the inputs u and g are never used by the algorithm. This means that a SealPacket ciphertext does not, in general, bind to the group's or user's identifiers. This could potentially allow a malicious user to impersonate another group member. When SealPacket is used within BoxMessage, it is always invoked on a message that contains the group's and the user's identifier, so the lack of group/user binding in SealPacket is not consequential there.

Nonce Repetition in Keybase. XSalsa20-Poly1305 is not secure when nonces repeat so our security analysis disallows nonce repetition between BoxMessage and/or SealPacket. The Keybase implementation uses uniformly random nonces, making collisions highly unlikely. Moreover, our results show that BoxMessage is robust to accidental non-uniformity in randomness as long nonces do not repeat. The XSalsa20-Poly1305 authenticated encryption scheme combines the XSalsa20 stream cipher and the Poly1305 *one-time* message authentication code. The stream is derived from the key and nonce and is used for keying Poly1305, so if nonces repeat then privacy and integrity may both be broken.

Message Encryption Scheme BM. Our modular symmetric signcryption construction BM models the BoxMessage chat-encryption algorithm as follows.

Construction 1. *Let* $\mathcal{M} \subseteq \{0,1\}^*$. *Let* NE *be a nonce-based encryption scheme. Let* H *be a hash function. Let* SP *be a deterministic symmetric signcryption scheme. Then* $\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}]$ *is the deterministic symmetric signcryption scheme as defined in Fig. 10, with message space* $\mathsf{BM.MS} = \mathcal{M}$ *and associated-data space* $\mathsf{BM.AD} = \{0,1\}^*$. *We require the following. The group key taken by* BM *is used as the key for both* NE *and* SP, *so* $\mathsf{BM.gkl} = \mathsf{NE.kl} = \mathsf{SP.gkl}$. *The nonce taken by* BM *is a pair containing a separate nonce for each of* NE *and* SP, *so* $\mathsf{BM.NS} = \{0,1\}^{\mathsf{NE.nl}} \times \mathsf{SP.NS}$.

```
$\underline{\mathsf{BM.UserKg}}$
$(sk, vk) \leftarrow\!\!\$\ \mathsf{SP.UserKg}$ ; Return $(sk, vk)$
$\underline{\mathsf{BM.SigEnc}(g, K_g, u, sk_u, n, m_{\mathsf{body}}, ad)}$            // $K_g \in \{0,1\}^{256}$
$(n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n$                                  // $n_{\mathsf{body}}, n_{\mathsf{header}} \in \{0,1\}^{192}$
// Encrypt the message body
$c_{\mathsf{body}} \leftarrow \mathsf{NE.Enc}(K_g, n_{\mathsf{body}}, m_{\mathsf{body}})$                        // $\mathsf{NE} = \mathsf{XSalsa20\text{-}Poly1305}$
// Create and encrypt the message header
$h_{\mathsf{body}} \leftarrow \mathsf{H}(n_{\mathsf{body}} \| c_{\mathsf{body}})$ ; $m_{\mathsf{header}} \leftarrow \langle ad, u, g, h_{\mathsf{body}} \rangle$  // $\mathsf{H} = \mathsf{SHA\text{-}256}$
$c_{\mathsf{header}} \leftarrow \mathsf{SP.SigEnc}(g, K_g, u, sk_u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon)$  // $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}$
Return $(c_{\mathsf{body}}, c_{\mathsf{header}})$
$\underline{\mathsf{BM.VerDec}(g, K_g, u, vk_u, n, c, ad)}$
$(n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n$ ; $(c_{\mathsf{body}}, c_{\mathsf{header}}) \leftarrow c$
// Recover and verify the message header
$m_{\mathsf{header}} \leftarrow \mathsf{SP.VerDec}(g, K_g, u, vk_u, n_{\mathsf{header}}, c_{\mathsf{header}}, \varepsilon)$
$h_{\mathsf{body}} \leftarrow \mathsf{H}(n_{\mathsf{body}} \| c_{\mathsf{body}})$
If $m_{\mathsf{header}} \neq \langle ad, u, g, h_{\mathsf{body}} \rangle$ then return $\perp$
// Recover and return the message body
$m_{\mathsf{body}} \leftarrow \mathsf{NE.Dec}(K_g, n_{\mathsf{body}}, c_{\mathsf{body}})$ ; Return $m_{\mathsf{body}}$
```

Fig. 10. Symmetric signcryption scheme $\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}]$. The right-aligned comments provide a guideline for modeling Keybase.

```
$\underline{\mathsf{SP.UserKg}}$
$(sk, vk) \leftarrow\!\!\$\ \mathsf{DS.Kg}$ ; Return $(sk, vk)$
$\underline{\mathsf{SP.SigEnc}(g, K_g, u, sk_u, n, m, ad)}$     // $K_g \in \{0,1\}^{256}$, $n \in \{0,1\}^{192}$, $ad = \varepsilon$
$h \leftarrow \mathsf{H}(m)$                              // $\mathsf{H} = \mathsf{SHA\text{-}512}$
$m_s \leftarrow$ “Keybase-Chat-2” $\| \langle K_g, n, h \rangle$
$s \leftarrow \mathsf{DS.Sig}(sk_u, m_s)$ ; $m_e \leftarrow s \| m$     // $\mathsf{DS} = \mathsf{Ed25519}$
$c \leftarrow \mathsf{NE.Enc}(K_g, n, m_e)$ ; Return $c$     // $\mathsf{NE} = \mathsf{XSalsa20\text{-}Poly1305}$
$\underline{\mathsf{SP.VerDec}(g, K_g, u, vk_u, n, c, ad)}$     // $ad = \varepsilon$
$m_e \leftarrow \mathsf{NE.Dec}(K_g, n, c)$
If $m_e = \perp$ then return $\perp$
$s \| m \leftarrow m_e$   // s.t. $|s| = \mathsf{DS.sl}$, $|m| \geq 0$
$h \leftarrow \mathsf{H}(m)$
$m_s \leftarrow$ “Keybase-Chat-2” $\| \langle K_g, n, h \rangle$
If $\neg\mathsf{DS.Ver}(vk_u, m_s, s)$ then return $\perp$ else return $m$
```

Fig. 11. Symmetric signcryption scheme $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{H}, \mathsf{DS}, \mathsf{NE}]$. The right-aligned comments provide a guideline for modeling Keybase.

Header Encryption Scheme SP. Our modular symmetric signcryption construction SP models the header-encryption algorithm SealPacket as follows.

Construction 2. *Let* H *be a hash function. Let* DS *be a deterministic digital signature scheme. Let* NE *be a nonce-based encryption scheme. Then* $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{H}, \mathsf{DS}, \mathsf{NE}]$ *is the symmetric signcryption scheme as defined in Fig. 11, with group-key length* $\mathsf{SP.gkl} = \mathsf{NE.kl}$*, nonce space* $\mathsf{SP.NS} = \{0,1\}^{\mathsf{NE.nl}}$*, message space* $\mathsf{SP.MS} = \{0,1\}^*$*, and associated-data space* $\mathsf{SP.AD} = \{\varepsilon\}$*.*

5 Security Analysis of Keybase Chat Encryption

In this section we analyze the security of the symmetric signcryption schemes BM and SP defined in Sect. 4. In Sect. 5.1, we show the in-group unforgeability of BM and SP. In Sects. 5.2 and 5.3, we show the out-group AE security of BM and SP. This requires us to introduce two weaker variants of the OAE security notion, one each for BM and SP, by relaxing the level of nonce-misuse requirements of the OAE game defined in Fig. 6. The SP analysis requires two new security notions, *$\mathcal{M}$-sparsity* for digital signature schemes and *authenticated encryption for key-dependent messages* for nonce-based encryption schemes.

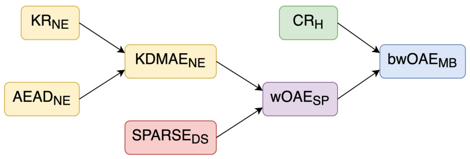
5.1 In-Group Unforgeability of BoxMessage and SealPacket

In-Group Unforgeability of BoxMessage. In-group unforgeability of $\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}]$ reduces to the security of SP and H. A BM ciphertext is a pair $(c_{\mathsf{body}}, c_{\mathsf{header}})$ comprising an NE ciphertext c_{body} and an SP ciphertext c_{header}, which encrypts $\langle ad, u, g, h_{\mathsf{body}}\rangle$. The adversary's objective is to forge a BM ciphertext by either forging c_{body} or c_{header}. The adversary can use a corrupt group key K_g, so c_{body} ciphertexts are easily forged. However, this does not suffice to produce a BM forgery because c_{header} encrypts the hash of c_{body}. Therefore, it would need to forge a corresponding c_{header} ciphertext. The IUF security of SP prevents the adversary from forging c_{header} ciphertexts. As a result, the adversary can only reuse honestly generated c_{header} from its prior queries to SigEnc in its forgery attempts. Since an honest c_{header} effectively commits to ad, u, g, h_{body}, and n_{header}, using an old c_{header} to construct a new BM ciphertext requires finding a new NE nonce-ciphertext pair that hashes to the same h_{body} under H. Collision resistance of H prevents this. The formal proof of Theorem 1 is in the full version.

Theorem 1. *Let* $\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}]$ *be the symmetric signcryption scheme built from some* $\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}$ *as specified in Construction 1. Let* $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ *and* $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicates as defined in Fig. 7. Let* $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}BM}}$ *be any adversary against the* IUF *security of* BM *with respect to* $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$*. Then we can build adversaries* $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$ *and* $\mathcal{A}_{\mathsf{CR}}$ *such that*

$$\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{BM},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}BM}}) \le \mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{SP},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}) + \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{CR}}).$$

In-Group Unforgeability of SealPacket. In-group unforgeability of the symmetric signcryption scheme $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{H}, \mathsf{DS}, \mathsf{NE}]$ reduces to the



$$\begin{tikzcd}
\mathsf{KR}_{\mathsf{NE}} \arrow[dr] & & \mathsf{CR}_{\mathsf{H}} \arrow[dr] & \\
& \mathsf{KDMAE}_{\mathsf{NE}} \arrow[dr] & & \mathsf{bwOAE}_{\mathsf{MB}} \\
\mathsf{AEAD}_{\mathsf{NE}} \arrow[ur] & & \mathsf{wOAE}_{\mathsf{SP}} \arrow[ur] & \\
& \mathsf{SPARSE}_{\mathsf{DS}} \arrow[ur] & &
\end{tikzcd}$$

Fig. 12. A summary of the reductions that we provide for the wOAE security of SP and the bwOAE security of BM.

security of DS and H. We parameterize the IUF security of SP to aim for a relaxed version of strong unforgeability because SP ciphertexts do not directly depend on the group's identifier g (even though it depends on the group key K_g).

An SP ciphertext encrypts $s \,\|\, m$ under K_g. The adversary can use a corrupt K_g, but forging an SP ciphertext still requires the signature s. So the adversary must either forge a new signature or reuse an honest signature from a prior SigEnc query. The SUFCMA security of DS prevents the former. An honest signature s is computed over "Keybase-Chat-2" $\|\, \langle K_g, n, h\rangle$ where h is the hash of the message m. Hence reusing an honest signature could use a new SP ciphertext that encrypts $s \,\|\, m$ with K_g, n, but the tidiness of NE prevents this. So reusing an honest signature requires finding a new message that hashes to the same h under H. Collision resistance of H prevents this. The formal proof of Theorem 2 is in the full version.

Theorem 2. *Let* SP = SEAL-PACKET-SS[H, DS, NE] *be the symmetric signcryption scheme built from some* H, DS, *and* NE *as specified in Construction 2. Let* $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 7. Let* $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$ *be any adversary against the* IUF *security of* SP *with respect to* $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}$. *Then we can build adversaries* $\mathcal{A}_{\mathsf{SUFCMA}}$ *and* $\mathcal{A}_{\mathsf{CR}}$ *such that*

$$\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{SP},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}) \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}}) + \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{CR}}).$$

5.2 Out-Group AE Security of BoxMessage

Out-group AE security of BM = BOX-MESSAGE-SS[$\mathcal{M}$, NE, H, SP] reduces to the security of its underlying primitives as summarized by the rightmost arrows of Fig. 12. At a high level, we show that BM achieves a variant of OAE security (bwOAE) if SP achieves another variant of OAE security (wOAE) and H is collision-resistant. Because NE = XSalsa20-Poly1305 in Keybase (which is not nonce-misuse resistant), both variants disallow nonce repetition.

Intuition. An BM ciphertext is a pair $(c_{\mathsf{body}}, c_{\mathsf{header}})$ consisting of an NE ciphertext c_{body} and an SP ciphertext c_{header}. One way the adversary could learn the challenge bit is by querying its VerDec oracle on a forged BM ciphertext that decrypts successfully. In order to accomplish that, the adversary must either forge the underlying SP ciphertext c_{header} or reuse an honestly generated c_{header}.

The former is prevented by the out-group AE security of SP. The latter is prevented by the collision resistance of H because of the following. An honestly generated c_{header} effectively commits to ad, u, g, h_{body}, and n_{header}. In order to reuse c_{header}, an adversary must find a new NE nonce-ciphertext pair that hashes to h_{body}, hence producing a collision. It follows that the VerDec oracle is essentially useless to the adversary; it can only serve to decrypt non-challenge ciphertexts that were previously returned by SigEnc. So it remains to show that the adversary cannot learn the challenge bit solely based on the BM ciphertexts that it receives from SigEnc. For any ciphertext $(c_{\mathsf{body}}, c_{\mathsf{header}})$ returned by SigEnc, the SP ciphertext c_{header} encrypts a hash of c_{body} but otherwise does not depend on the challenge bit. So the adversary gains no advantage from observing c_{header}. Finally, the AEAD security of NE guarantees that c_{body} does not reveal the challenge bit.

Because the header encryption scheme SP and the body encryption scheme NE use the same symmetric key K_g, we require integrity of SP ciphertexts produced using K_g hold even when the adversary can obtain other NE encryptions under the same key. Similarly, the NE ciphertexts generated using the symmetric key K_g should be indistinguishable even when the adversary can obtain SP encryptions and decryptions under the same key. We introduce a variant of the OAE game in Definition 3 to capture these joint requirements.

Restrictions on Nonce Misuse in BM and SP. We now define new variants of out-group AE security for our analysis of Keybase. The BM and SP schemes in Keybase are not nonce-misuse resistant so we modify the OAE game to disallow nonce repetition. We start with wOAE security for SP.

Definition 1. *Let* SS *be a symmetric signcryption scheme. Consider the* OAE *security game for* SS *of Fig. 6 (w.r.t. any* $\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}}$*,* $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$*). We define a new variant of this game as follows. The instruction preventing nonce misuse*

$$\mathsf{require}\ \forall d \in \{0,1\}, (g,u,n,m_d,ad) \notin N_d \quad \textit{is replaced with} \quad \mathsf{require}\ (g,n) \notin N.$$

In addition, the instructions updating the nonce set

$$\begin{matrix} N_0 \leftarrow N_0 \cup \{(g,u,n,m_0,ad)\} \\ N_1 \leftarrow N_1 \cup \{(g,u,n,m_1,ad)\} \end{matrix} \quad \textit{are replaced with} \quad N \leftarrow N \cup \{(g,n)\}.$$

We denote the resulting game (and security notion) by wOAE*. It is a weak variant of* OAE *that does not require nonce-misuse resistance. We define an adversary's advantage in breaking the* wOAE *security of* SS *in the natural way.*

Now we define bwOAE security for BM. The nonce of BM is a pair of two separate nonces $n = (n_{\mathsf{body}}, n_{\mathsf{header}})$. The bwOAE security game independently applies the group-nonce uniqueness condition introduced in Definition 1 to each of (g, n_{body}) and (g, n_{header}), *and* it also requires that $n_{\mathsf{body}} \neq n_{\mathsf{header}}$. This is a necessary because BM calls $\mathsf{NE.Enc}$ on (g, n_{body}), and SP calls $\mathsf{NE.Enc}$ on (g, n_{header}). In Keybase both NE schemes are $\mathsf{XSalsa20\text{-}Poly1305}$ using the same key.

Definition 2. *Let* $\mathcal{X}, \mathcal{Y}$ *be any sets. Let* SS *be a symmetric signcryption scheme with the nonce space* $\mathsf{SS.NS} = \mathcal{X} \times \mathcal{Y}$*. Consider the* OAE *security game for* SS *of Fig. 6 (w.r.t. any* $\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}}$*,* $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$*). We define a new variant of this game as follows. The instruction preventing nonce misuse*

$\mathsf{require}\ \forall d, (g, u, n, m_d, ad) \notin N_d$ *is replaced with*

```
$(n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n$
If $n_{\mathsf{body}} = n_{\mathsf{header}}$ then return $\bot$
If $(g, n_{\mathsf{body}}) \in N$ then return $\bot$
If $(g, n_{\mathsf{header}}) \in N$ then return $\bot$.
```

In addition, the instructions updating the nonce set

```
$N_0 \leftarrow N_0 \cup \{(g, u, n, m_0, ad)\}$
$N_1 \leftarrow N_1 \cup \{(g, u, n, m_1, ad)\}$
```

are replaced with

```
$N \leftarrow N \cup \{(g, n_{\mathsf{header}})\}$
$N \leftarrow N \cup \{(g, n_{\mathsf{body}})\}$.
```

We denote the resulting game (and security notion) by bwOAE. *Beyond being defined for* SS *with a bipartite nonce space, this variant of* OAE *is weak in that it does not require nonce-misuse resistance. We define an adversary's advantage in breaking the* bwOAE *security of* SS *in the natural way.*

The Joint Security Required of SP and NE. Here we define the security notion required from SP when it is used in the presence of arbitrary NE encryptions under the same symmetric group keys that are used by SP. We call this notion wOAE[ENC[$\mathcal{M}$, NE]]. It is a parameterized version of the wOAE game defined in Definition 1. We use it for our analysis of the bwOAE security of BM.

At the start of this section we discussed that the security reduction for BM intuitively requires that it is hard to forge an SP ciphertext (without knowing the corresponding group key K_g) in the presence of NE encryptions. Our definition of wOAE[ENC[$\mathcal{M}$, NE]] captures this by providing the adversary access to an NE encryption oracle ENC in addition to the SIGENC and VERDEC oracles in the out-group AE security game of SP. We stress that proving the security of BM does not, in principle, require us to provide the SIGENC oracle to the adversary. We choose to require this stronger level of security from SP because of the following reasons. On the one hand, in Sect. 4 we explained why it is beneficial to prove that SP satisfies a strong security notion, going beyond what is required by BM. On the other hand, this stronger security notion that we require from SP will not come at the cost of introducing additional assumptions or achieving looser concrete-security bounds in our analysis of BM.

Definition 3. *Let* SS *be a symmetric signcryption scheme. Let $\mathcal{M} \subseteq \{0,1\}^*$. Let* NE *be a nonce-based encryption scheme. Consider the* wOAE *security game for* SS *as defined in Definition 1 (w.r.t. any $\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}}$, $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$). We define a variant of this game by adding an oracle that is defined as follows.*

```
$\underline{\text{ENC}[\mathcal{M}, \mathsf{NE}](g, n_{\mathsf{body}}, m_{\mathsf{body},0}, m_{\mathsf{body},1})}$
require $\mathsf{K}[g] \neq \bot$ and $|m_{\mathsf{body},0}| = |m_{\mathsf{body},1}|$
require $(g, n_{\mathsf{body}}) \notin N$ and $m_{\mathsf{body},0}, m_{\mathsf{body},1} \in \mathcal{M}$
If $m_{\mathsf{body},0} \neq m_{\mathsf{body},1}$ then
    If $\mathsf{group\_is\_corrupt}[g]$ then return $\bot$
    $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$c_{\mathsf{body}} \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n_{\mathsf{body}}, m_{\mathsf{body},b})$
$N \leftarrow N \cup \{(g, n_{\mathsf{body}})\}$ ; Return $c_{\mathsf{body}}$
```

It shares set N, bit b, and the tables K, group_is_corrupt, chal *with the rest of the security game. We denote the resulting game (and security notion) by* wOAE[ENC[$\mathcal{M}$, NE]]. *It simultaneously requires out-group AE security of* SS *(without nonce repetition) and an IND-style security of* NE*. We define an adversary's advantage in breaking this security notion in the natural way.*

Note that we require the messages that the adversary queries to the ENC oracle to be in $\mathcal{M}$. Intuitively, in our security analysis of BM, an adversary will only be able to obtain NE encryptions of messages in the message space of BM. So in the security reduction for BM we will use $\mathcal{M} = \mathsf{BM.MS}$.

Out-Group AE Security of BoxMessage. We prove bwOAE security of BM. The formal proof of Theorem 3 is in the full version.

Theorem 3. *Let* $\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}]$ *be the symmetric signcryption scheme built from some* $\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}$ *as specified in Construction 1. Let* $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ *and* $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicates as defined in Fig. 7. Let* $\mathsf{func}^{\perp}_{\mathsf{out}}$ *be the output-guarding functions as defined in Fig. 8. Let* $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ *be the security notion as defined in Definition 3. Let* $\mathcal{A}_{\mathsf{bwOAE\text{-}of\text{-}BM}}$ *be any adversary against the* bwOAE *security of* BM *with respect to* $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ *and* $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$*. Then we build adversaries* $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ *and* $\mathcal{A}_{\mathsf{CR}}$ *such that*

$$\mathsf{Adv}^{\mathsf{bwOAE}}_{\mathsf{BM},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}},\mathsf{func}^{\perp}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{bwOAE\text{-}of\text{-}BM}}) \leq \mathsf{Adv}^{\mathsf{wOAE}[\textsc{Enc}[\mathcal{M},\mathsf{NE}]]}_{\mathsf{SP},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}},\mathsf{func}^{\perp}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}) + \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{CR}}).$$

Note that we prove the security of BM with respect to $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ and $\mathsf{func}^{\perp}_{\mathsf{out}}$. As discussed in Sect. 3, $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ essentially requires BM to have ciphertext integrity. Our result relies on the security of SP with respect to $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ and $\mathsf{func}^{\perp}_{\mathsf{out}}$. Recall that $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ basically requires SP to have ciphertext integrity, except it allows for an honest ciphertext to be successfully decrypted even with respect to a wrong user identifier; the latter is not considered a "valid" forgery. This does not translate to an attack against BM because it only uses SP to encrypt header messages $m_{\mathsf{header}} = \langle ad, u, g, h_{\mathsf{body}} \rangle$ that contain u, and the $\mathsf{BM.VerDec}$ algorithm verifies that the group identifier it received as input matches the one that was parsed from m_{header}.

5.3 Out-Group AE Security of SealPacket

Out-group AE security of $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{H}, \mathsf{DS}, \mathsf{NE}]$ reduces to the security NE and DS (see Fig. 12). In particular, $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ security holds if NE provides authenticated encryption for key-dependent messages and DS produces $\mathcal{M}$-sparse signatures. We introduce these security notions below.

Intuition. Recall that in the $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ game, the adversary is provided with (un)signcryption oracles SIGENC and VERDEC for SP, and an encryption oracle ENC for NE. Each of these returns output based on a challenge bit that is shared between them. The adversary can use three approaches to learn the challenge bit. It can (a) attempt SP forgeries by calling its SP decryption oracle VerDec; (b) make left-or-right queries to its NE encryption oracle Enc; (c) make left-or-right queries to its SP encryption oracle SigEnc.

The adversary is allowed to expose users' signing keys so it could attempt to forge an SP ciphertext using an exposed DS signing key and its Enc oracle. The adversary would then query the resulting ciphertext to its VerDec oracle in an attempt to trivially win the game. We show that the adversary is unable to accomplish this. The Enc oracle is defined to only produce encryptions of the messages from the set $\mathcal{M}$. In the implementation of Keybase, the messages from $\mathcal{M}$ have a specific encoding; we will rely on this property in our proof. In contrast, any ciphertext successfully decrypted by VerDec must encrypt a message of the form $m_e = s \,\|\, m$ where s is a valid DS signature. So the adversary needs to find a signature s that is consistent with the message encoding that is permitted by Enc. The $\mathcal{M}$-sparseness of DS signatures, which we formalize below, prevents this. It follows that the VerDec oracle does not help the adversary to win the game by querying ciphertexts that were previously returned by Enc.[1]

Now we can reimagine the Enc and SigEnc oracles as producing NE encryptions of key-dependent messages. The SigEnc oracle requires messages to be derived as a specific function of the symmetric group key K_g. The Enc oracle can be thought of as allowing to messages that are derived from "constant" functions, meaning the chosen messages do not depend on K_g. We can also view the VerDec oracle as an NE decryption oracle that prevents the adversary from trivially winning the game by merely querying the ciphertexts it previously obtained from either Enc or SigEnc. We define the AE security of NE for key-dependent messages and show that the adversary can only win the $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ game against SP if it can win the KDMAE game against NE.

Reliance on the Message Encoding in Keybase. We mentioned in the intuition that we rely on the encoding of messages in $\mathcal{M}$ in our proof. We emphasize that avoiding this dependency is non-trivial. The cyclic key dependency within SP and the key reuse between BM and SP pose significant challenges when considering the possibility of an alternate proof.

$\mathcal{M}$-sparse Signatures. Consider game $\mathcal{G}^{\mathsf{SPARSE}}$ of Fig. 13, defined for a digital signature scheme DS, a set $\mathcal{M} \subseteq \{0,1\}^*$, and an adversary $\mathcal{A}_{\mathsf{SPARSE}}$. The advantage of $\mathcal{A}_{\mathsf{SPARSE}}$ in breaking the $\mathcal{M}$-SPARSE security of DS is defined as $\mathsf{Adv}^{\mathsf{SPARSE}}_{\mathsf{DS},\mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}}) = \Pr[\mathcal{G}^{\mathsf{SPARSE}}_{\mathsf{DS},\mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}})]$. Intuitively, this game captures the inability of an adversary to produce a signature that conforms to the message space $\mathcal{M}$ even though the adversary chooses the public key used to verify the signature. More formally, the adversary wins if it is able to return (vk, m, s, γ) such that s verifies as a signature over the message m under the verification key vk and $s \,\|\, \gamma \in \mathcal{M}$. We stress that the adversary is allowed to choose an arbitrary – possibly malformed – verification key. The adversary is not required to know the corresponding signing key, and such a key may in fact not exist.

[1] The $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ game itself also prevents the adversary from trivially winning by querying VerDec on a ciphertext that was previously returned by SigEnc.

We verify our intuition about the $\mathcal{M}$-sparsity of the $\mathsf{Ed25519}$ signature scheme underlying SP in the full version. $\mathsf{Ed25519}$ is a deterministic signature scheme introduced by Bernstein, Duif, Lange, Schwabe, and Yang in [17]. It is obtained by applying the commitment-variant of the Fiat-Shamir transform to an identification scheme. Therefore a signature produced by $\mathsf{Ed25519}$ consists of the commitment and response of the identification scheme. The adversary can only win the SPARSE game of $\mathsf{Ed25519}$ if it is able to produce an accepting conversation transcript for the identification scheme such that the corresponding commitment conforms to $\mathcal{M}$. Commitments in the identification scheme underlying $\mathsf{Ed25519}$ are elements of a prime-order group. We prove that finding such a commitment is only possible if the adversary is able to find a group element and its discrete logarithm such that the group element is in $\mathcal{M}$ which we show is hard in the generic group model.

```
$\mathcal{G}^{\mathsf{SPARSE}}_{\mathsf{DS},\mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}})$
$(vk, m, s, \gamma) \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{SPARSE}}$
$\mathsf{win}_0 \leftarrow \mathsf{DS.Ver}(vk, m, s)$
$\mathsf{win}_1 \leftarrow (s \,\|\, \gamma \in \mathcal{M})$
Return $\mathsf{win}_0$ and $\mathsf{win}_1$
```

Fig. 13. Game defining $\mathcal{M}$-sparsity of a digital signature scheme DS for a set $\mathcal{M}$.

The Message Space $\mathcal{M}$. Keybase uses the MessagePack serialization format [21] to encode plaintext messages. Plaintext messages are represented using a custom data structure in Keybase. So the serialized MessagePack encoding of a plaintext is a byte sequence that not only stores the plaintext itself but also some metadata about the data structure that represents it. For messages encrypted by BM, the metadata about the data structure happens be located in the first 17 bytes of the encoding. This means that the encoding of every plaintext encrypted by BM contains a fixed 17-byte prefix. Let this 17-byte prefix be pre. Then we define the message space of BM by $\mathsf{BM.MS} = \{\mathsf{pre} \,\|\, \nu \mid \nu \in \{0,1\}^*\}$.

Message-Deriving Functions. Let ϕ be any function that takes a symmetric key K as input and uses it to derive and return some message m. We call ϕ a *message-deriving* function and will consider some classes (i.e. sets) $\varPhi$ of message-deriving functions. We require that the length of an output returned by ϕ must not depend on its input; we denote the output length of ϕ by $\|\phi\|$.

AE Security of NE for Key-Dependent Messages. Consider game $\mathcal{G}^{\mathsf{KDMAE}}$ of Fig. 14, defined for a nonce-based encryption scheme NE, a class of message-deriving functions $\varPhi$, and an adversary $\mathcal{A}_{\mathsf{KDMAE}}$. The advantage of $\mathcal{A}_{\mathsf{KDMAE}}$ in breaking the $\varPhi$-KDMAE security of NE is defined as $\mathsf{Adv}^{\mathsf{KDMAE}}_{\mathsf{NE},\varPhi}(\mathcal{A}) = 2 \cdot \Pr[\mathcal{G}^{\mathsf{KDMAE}}_{\mathsf{NE},\varPhi}(\mathcal{A})] - 1$. This game can be thought of as a modification of the AEAD security game for NE (Fig. 4) which does not require nonce-misuse resistance. The core difference is that the $\textsc{Enc}$ oracle now takes message-deriving functions $\phi_0, \phi_1 \in \varPhi$ as input. The challenge message is derived as $m_b \leftarrow \phi_b(\mathsf{K}[g])$ for $b \in \{0,1\}$, where $\mathsf{K}[g]$ is the symmetric group key associated to g. Trivial attacks are prevented by requiring that ϕ_0, ϕ_1 have the same output length and that $\phi_0 = \phi_1$ whenever $\textsc{Enc}$ is called for a corrupt group. Our definition is based on prior work [6,12,13,19]. There are strong impossibility results [12] regarding

```
Game $\mathcal{G}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi}(\mathcal{A}_{\mathsf{KDMAE}})$
$b \leftarrow\$ \{0,1\}$ ; $b' \leftarrow\$ \mathcal{A}^{\mathrm{G,E{\scriptsize NC},D{\scriptsize EC}}}_{\mathsf{KDMAE}}$
Return $b = b'$

$\textsc{Dec}(g, n, c)$
require $\mathsf{K}[g] \neq \bot$ and $\neg\mathsf{group\_is\_corrupt}[g]$
require $(g, n, c) \notin C$
$m \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c)$
If $b = 0$ then return $\bot$ else return $m$

$\textsc{Enc}(g, n, \phi_0, \phi_1)$
require $\mathsf{K}[g] \neq \bot$ and $(g, n) \notin N$
require $\phi_0, \phi_1 \in \Phi$ and $\|\phi_0\| = \|\phi_1\|$
If $\phi_0 \neq \phi_1$ then
    If $\mathsf{group\_is\_corrupt}[g]$ then return $\bot$
    $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$m_b \leftarrow \phi_b(\mathsf{K}[g])$ ; $c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_b)$
$N \leftarrow N \cup \{(g, n)\}$ ; $C \leftarrow C \cup \{(g, n, c)\}$
Return $c$

$\textsc{NewHonGroup}(g)$
require $\mathsf{K}[g] = \bot$
$\mathsf{K}[g] \leftarrow\$ \{0,1\}^{\mathsf{NE.kl}}$

$\textsc{ExposeGroup}(g)$
require $\mathsf{K}[g] \neq \bot$ and $\neg\mathsf{chal}[g]$
$\mathsf{group\_is\_corrupt}[g] \leftarrow \mathsf{true}$
Return $\mathsf{K}[g]$

$\textsc{NewCorrGroup}(g, K)$
require $\mathsf{K}[g] = \bot$
$\mathsf{group\_is\_corrupt}[g] \leftarrow \mathsf{true}$
$\mathsf{K}[g] \leftarrow K$
```

Fig. 14. Game defining authenticated-encryption security of NE for Φ-key-dependent messages, where Φ is a class of message-deriving functions and G = {NewHonGroup, ExposeGroup, NewCorrGroup}.

the existence of schemes that are secure with respect to very large classes of message-deriving functions Φ. We sidestep these results by considering a very narrow and simple class Φ_{SP} that we define below.

The Class of Message-Deriving Functions Φ_{Sp} Earlier we discussed that in the wOAE[Enc[$\mathcal{M}$, NE]] security game for SP, the SigEnc and Enc oracles can be thought of as returning an NE ciphertext that encrypts an output of some message-deriving function. We now define the class Φ_{SP} containing all message-deriving functions that are used by either SigEnc or Enc.

Construction 3. *Let* NE *be a nonce-based encryption scheme. Let* H *be a hash function. Let* DS *be a digital signature scheme. Let* SIGENC-DER *and* ENC-DER *be the parameterized message-deriving functions that are defined as follows, each taking an* NE *key* $K \in \{0,1\}^{\mathsf{NE.kl}}$ *as input.*

```
$\mathsf{SIGENC\text{-}DER}[\mathsf{NE}, \mathsf{H}, \mathsf{DS}, m, n, sk](K)$
$h \leftarrow \mathsf{H}(m)$ ; $m_s \leftarrow$ "Keybase-Chat-2" $\|\ \langle K, n, h\rangle$
$s \leftarrow \mathsf{DS.Sig}(sk, m_s)$ ; $m_e \leftarrow s \,\|\, m$ ; Return $m_e$

$\mathsf{ENC\text{-}DER}[m](K)$
Return $m$
```

Then $\Phi_{\mathsf{SP}} = \mathsf{MSG\text{-}DER\text{-}FUNC}[\mathsf{NE}, \mathsf{H}, \mathsf{DS}]$ *is the class of all message-deriving functions of these forms.*

Note that SIGENC-DER only uses K as a part of the message m_s signed by DS.Sig. Keybase instantiates DS with Ed25519 which computes two SHA-512 hashes of m_s (mixed with other inputs). The resulting signature does not depend on m_s in any other way. Using this observation and an indifferentiability result of Bellare, Davis, and Di [10] (for SHA-512 with output reduced modulo a prime) we capture SIGENC-DER as a special class of message-deriving functions for which we can prove security in the random oracle model.

KDMAE Security for Messages Derived from a Hashed Key. Let H be a hash function. Let Φ be a class of message-deriving functions such that each $\phi \in \Phi$ on input K is only allowed to derive messages from the hash value $\mathsf{H}(K)$, and never directly from K. We will roughly show that every AEAD-secure nonce-based encryption scheme NE is also Φ-KDMAE-secure, provided that H is modeled as a random oracle. We formalize this class of functions as follows.

Definition 4. *We say Φ* derives messages from a hashed key *if there exists a set Γ and a function H (modeled as a random oracle) such that $\Phi = \{\phi_\gamma \mid \phi_\gamma(\cdot) = \gamma(\mathsf{H}(\cdot)), \gamma \in \Gamma\}$.*

In the full version we show how to capture Φ_{SP} as satisfying this definition. Thereby, the following result will give us Φ_{SP}-KDMAE security.

Proposition 1. *Let NE be a nonce-based encryption scheme. Let Φ be a class of message-deriving functions that derives messages from a hash key. Let $\mathcal{A}_{\mathsf{KDMAE}}$ be an adversary against the Φ-KDMAE security of NE making $q_{\textsc{NewHonGroup}}$ queries to its $\textsc{NewHonGroup}$ oracle. Then we can build adversaries $\mathcal{A}_{\mathsf{KR}}$ and $\mathcal{A}_{\mathsf{AEAD}}$ such that (in the random oracle model)*

$$\mathsf{Adv}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi}(\mathcal{A}_{\mathsf{KDMAE}}) \leq 2 \cdot \mathsf{Adv}^{\mathsf{KR}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{KR}}) + \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}}) + \frac{q^2_{\textsc{NewHonGroup}}}{2^{\mathsf{NE.kl}}}.$$

The constructed adversaries will not repeat (g, n) across $\textsc{Enc}$ queries, so non-nonce-misuse resistant NE suffices. To prove this, we first assert that a Φ-KDMAE adversary $\mathcal{A}_{\mathsf{KDMAE}}$ can never directly query the random oracle on any of the (non-exposed) honest keys; otherwise, we could use $\mathcal{A}_{\mathsf{KDMAE}}$ in order to break the key-recovery security of AEAD. But then $\mathcal{A}_{\mathsf{KDMAE}}$ cannot distinguish between messages derived from $\mathsf{H}(K[g])$ or from some $\mathsf{H}^*(g)$. Here H is the actual random oracle and H^* is a simulated random oracle whose output depends on a group's identifier g instead of this group's key $K[g]$. We switch from using $\mathsf{H}(K[g])$ to $\mathsf{H}^*(g)$, thus breaking the dependency of each challenge message on the corresponding NE key. The AEAD security of NE then guarantees that $\mathcal{A}_{\mathsf{KDMAE}}$ cannot guess the challenge bit. The formal proof of Proposition 1 is in the full version.

Out-Group AE Security of $\mathsf{SealPacket}$. We prove wOAE security of SP. The formal proof of Theorem 4 is in the full version.

Theorem 4. *Let $\mathcal{M} \subseteq \{0,1\}^*$. Let $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{H}, \mathsf{DS}, \mathsf{NE}]$ be the symmetric signcryption scheme built from some $\mathsf{H}, \mathsf{DS}, \mathsf{NE}$ as specified in Construction 2. Let $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ be the ciphertext-triviality predicate as defined in Fig. 7. Let $\mathsf{func}^{\perp}_{\mathsf{out}}$ be the output-guarding function as defined in Fig. 8. Let $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ be the security notion as defined in Definition 3. Let $\Phi_{\mathsf{SP}} = \mathsf{MSG\text{-}DER\text{-}FUNC}[\mathsf{NE}, \mathsf{H}, \mathsf{DS}]$ be the class of message-deriving functions defined in Construction 3. Let $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ be an adversary against the $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ security of SP with respect to $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ and $\mathsf{func}^{\perp}_{\mathsf{out}}$. Then we can build adversaries $\mathcal{A}_{\mathsf{KDMAE}}$ and $\mathcal{A}_{\mathsf{SPARSE}}$ such that*

$$\mathsf{Adv}^{\mathsf{wOAE}[\textsc{Enc}[\mathcal{M},\mathsf{NE}]]}_{\mathsf{SP},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}},\mathsf{func}^{\perp}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}) \leq \mathsf{Adv}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi_{\mathsf{SP}}}(\mathcal{A}_{\mathsf{NE}}) + 2 \cdot \mathsf{Adv}^{\mathsf{SPARSE}}_{\mathsf{DS},\mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}}).$$

The OAE security results in Theorems 3 and 4 used the weaker output guarding function $\mathsf{func}_{\mathsf{out}}^{\perp}$. In the full version of this paper, we show that for $\mathsf{SS} \in \{\mathsf{BM}, \mathsf{SP}\}$, the OAE security of SS with respect to $\mathsf{func}_{\mathsf{out}}^{\perp}$ implies its OAE security with respect to the stronger output guarding function $\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}$.

6 Conclusions

Combining Theorem 1 with Theorem 2 and Theorem 3 with Theorem 4 establishes the in-group unforgeability and out-group authenticated encryption security of Keybase's BoxMessage algorithm. These results rely on some standard security assumptions (unforgeability of Ed25519 and collision resistance of SHA-256) as well as some non-standard assumptions (key-dependent message security of XSalsa20-Poly1305 and sparsity of Ed25519). These non-standard assumptions arose, respectively, from the key cycle in SealPacket and the key reuse without explicit context separation BoxMessage. While we were able to justify these assumptions, we consider them brittle as they are not well studied, their justifications required ideal models, and (in the case of sparsity) they required properties of the specific messaging encoding format used by Keybase.

The comparative simplicity of our Sign-then-Encrypt construction speaks to the value of formalizing the syntax and security of symmetric signcryption. Explicit goals allow designing schemes in parallel with writing proofs to identify precisely what is needed.

References

1. Albrecht, M., Dowling, B., Jones, D.: Device-oriented group messaging: a formal cryptographic analysis of matrix'core. In: IEEE S&P 2024 (2023)
2. Albrecht, M.R., Celi, S., Dowling, B., Jones, D.: Practically-exploitable cryptographic vulnerabilities in matrix. In: 2023 IEEE Symposium on Security and Privacy (SP), pp. 1419–1436. IEEE Computer Society (2022)
3. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Modular design of secure group messaging protocols and the security of MLS. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021, pp. 1463–1483. ACM Press, November 2021. https://doi.org/10.1145/3460120.3484820
4. Alwen, J., Janneck, J., Kiltz, E., Lipp, B.: The pre-shared key modes of HPKE. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology - ASIACRYPT 2023. Springer, Heidelberg (2023). https://doi.org/10.1007/978-981-99-8736-8_11
5. An, J.H., Dodis, Y., Rabin, T.: On the security of joint signature and encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 83–107. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_6
6. Backes, M., Pfitzmann, B., Scedrov, A.: Key-dependent message security under active attacks - BRSIM/UC-soundness of symbolic encryption with key cycles. In: Sabelfeld, A. (ed.) CSF 2007 Computer Security Foundations Symposium, pp. 112–124. IEEE Computer Society Press (2007). https://doi.org/10.1109/CSF.2007.23
7. Balbás, D., Collins, D., Gajland, P.: WhatsUpp with sender keys? Analysis, improvements and security proofs. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology - ASIACRYPT 2023, pp. 307–341. Springer, Heidelberg (2023). https://doi.org/10.1007/978-981-99-8733-7_10

8. Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., Cohn-Gordon, K.: The Messaging Layer Security (MLS) Protocol. RFC 9420, July 2023. https://doi.org/10.17487/RFC9420
9. Barnes, R., Bhargavan, K., Lipp, B., Wood, C.A.: Hybrid Public Key Encryption. RFC 9180, February 2022. https://doi.org/10.17487/RFC9180
10. Bellare, M., Davis, H., Di, Z.: Hardening signature schemes via derive-then-derandomize: stronger security proofs for EdDSA. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 223–250. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-31368-4_9
11. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: 38th FOCS, pp. 394–403. IEEE Computer Society Press, October 1997. https://doi.org/10.1109/SFCS.1997.646128
12. Bellare, M., Keelveedhi, S.: Authenticated and misuse-resistant encryption of key-dependent data. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 610–629. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_35
13. Bellare, M., Meiklejohn, S., Thomson, S.: Key-versatile signatures and applications: RKA, KDM and joint Enc/Sig. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 496–513. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_28
14. Bellare, M., Stepanovs, I.: Security under message-derived keys: Signcryption in iMessage. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 507–537. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-45727-3_17
15. Bernstein, D.J.: The Poly1305-AES message-authentication code. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 32–49. Springer, Heidelberg (2005). https://doi.org/10.1007/11502760_3
16. Bernstein, D.J.: The Salsa20 family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 84–97. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3_8
17. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 124–142. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23951-9_9
18. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. J. Cryptogr. Eng. **2**(2), 77–89 (2012). https://doi.org/10.1007/s13389-012-0027-1
19. Black, J., Rogaway, P., Shrimpton, T.: Encryption-scheme security in the presence of key-dependent messages. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 62–75. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36492-7_6
20. Brendel, J., Cremers, C., Jackson, D., Zhao, M.: The provable security of Ed25519: theory and practice. In: 2021 IEEE Symposium on Security and Privacy, pp. 1659–1676. IEEE Computer Society Press, May 2021. https://doi.org/10.1109/SP40001.2021.00042
21. Furuhashi, S.: Messagepack. https://msgpack.org/
22. Keybase: Keybase Book. https://book.keybase.io/
23. Keybase: Keybase Book—Chat—Crypto. https://github.com/keybase/book-content/blob/master/D-docs/04-chat/01-crypto.md?plain=1#L89-L93
24. Keybase: Keybase client. https://github.com/keybase/client
25. Keybase: Keybase client—boxer.go—BoxMessage. https://github.com/keybase/client/blob/v6.2.2/go/chat/boxer.go/#L1564-L1566

26. Keybase: Keybase client—codec.go—Design Notes. https://github.com/keybase/client/blob/v6.2.2/go/chat/signencrypt/codec.go/#L95-L110
27. Keybase: Keybase stats. https://web.archive.org/web/20200207065125/https://keybase.io/. Accessed 28 Feb 2024
28. Marlinspike, M.: Private group messaging, May 2014. https://signal.org/blog/private-groups/
29. Rogaway, P.: Nonce-based symmetric encryption. In: Roy, B.K., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 348–359. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-25937-4_22
30. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_23
31. Ryan, K., Pornin, T., Fitzgerald, S.: Keybase protocol security review, February 2019. https://keybase.io/docs-assets/blog/NCC_Group_Keybase_KB2018_Public_Report_2019-02-27_v1.3.pdf
32. Shrimpton, T.: A characterization of authenticated-encryption as a form of chosen-ciphertext security. Cryptology ePrint Archive, Report 2004/272 (2004). https://eprint.iacr.org/2004/272
33. Wallez, T., Protzenko, J., Beurdouche, B., Bhargavan, K.: TreeSync: authenticated group management for messaging layer security. In: 32nd USENIX Security Symposium, pp. 1217–1233. USENIX Association, Anaheim, CA, August 2023
34. WhatsApp: Whatsapp encryption overview: Technical white paper, September 2023. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf
35. Zheng, Y.: Digital signcryption or how to achieve cost(signature & encryption) $\ll$ cost(signature) + cost(encryption). In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 165–179. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052234
36. Zoom: Zoom acquires keybase and announces goal of developing the most broadly used enterprise end-to-end encryption offering, May 2020. https://blog.zoom.us/zoom-acquires-keybase-and-announces-goal-of-developing-the-most-broadly-used-enterprise-end-to-end-encryption-offering/