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, Georgia, US
{josephjaeger,akshayakumar}@gatech.edu
https://cc.gatech.edu/~josephjaeger/
https://cc.gatech.edu/~akumar805/
[2] igors.stepanovs@gmail.com
https://igors.org/

23 May 2024

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.

Contents

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 [Keyi]. In May 2020, Keybase was acquired by Zoom. At the time, Zoom issued a public statement [Zoo20] 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 the 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 [Keyc], but the server is not. Our security analysis primarily relies on the source code. Keybase also provides the "Keybase Book" website [Keya] with excellent documentation that explains its cryptographic design. The only prior security analysis of Keybase was done by the NCC Group in 2019 [RPF19], 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 [BCG23,Mar14] 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* [Mar14], *WhatsApp* [Wha23], and *Matrix* [ACDJ22,ADJ23] messengers. In addition, the *Messaging Layer Security* (MLS) [BBR+23] protocol contains a component called *FS-GAEAD* [ACDT21] or *TreeDEM* [WPBB23] 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 where every symmetric key is known to all group members. In such a protocol, group members can 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)

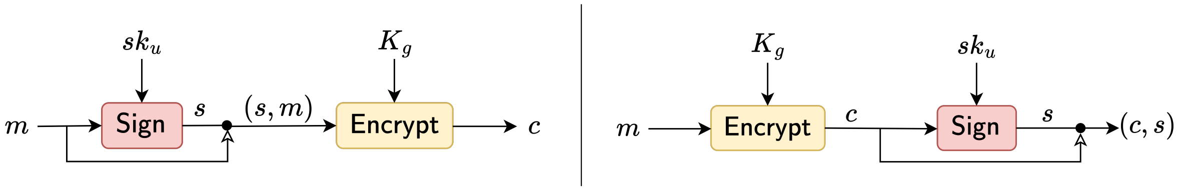

Fig. 1: Warmup schemes obtained by composing digital signatures with symmetric encryption. **Left pane:** Sign-then-Encrypt. **Right pane:** Encrypt-then-Sign.

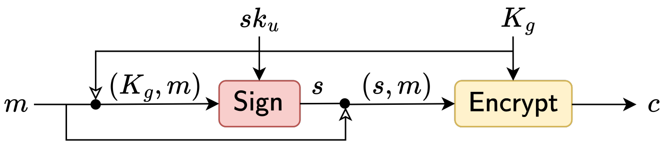

Fig. 2: A high-level representation of the SealPacket scheme in Keybase.

to obtain and return a symmetric ciphertext 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 [ADR02], which is a standard cryptographic primitive that combines digital signatures with public-key encryption [Zhe97], 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 [ADR02]. 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 [BCG23] prescribes to sign a symmetric ciphertext; and this is indeed done by the *Signal*, *WhatsApp*, and *Matrix* messengers. The MLS protocol [BBR$^+$23] protocol prescribes encrypting 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 [Keye]:

> *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 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 chatbot (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).

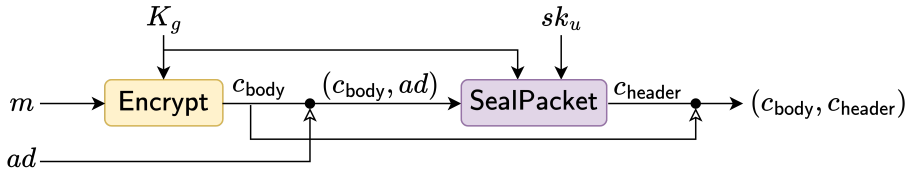

Fig. 3: A high-level representation of the BoxMessage scheme in Keybase.

Keybase uses the auxiliary-data field ad to authenticate a metadata header for the chat message. This header contains the group's identity and the 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) [Keyb]:

> *The 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.

Note that the use of sender-specific encryption keys in the *Sender Keys* [BCG23] and MLS [BBR+23] 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 the 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 [Keyf] and BoxMessage is implemented in [Keyd]. These schemes are instantiated with the nonce-based authenticated encryption scheme XSalsa20-Poly1305 [Ber05,Ber08] and the digital signature scheme Ed25519 [BDL+11,BDL+12]. 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 both 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 [BDJR97], 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 our 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 [BCJZ21,BDD23].

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) scheme is specified in RFC 9180 [BBLW22]. Alwen, Janneck, Kiltz, and Lipp [AJKL23] 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 [AJKL23] use one more set of keys, and the definitions in [AJKL23] 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

Basic Notation. Let $\mathbb{N} = \{0, 1, 2, \ldots\}$. We denote the empty string by ε. If $x \in \{0,1\}^*$ is a bit-string then $|x|$ denotes its length, $x[i]$ denotes its i-th bit for $1 \le i \le |x|$, and $x[i..j] = x[i] \ldots x[j]$ for $1 \le i \le j \le |x|$. By $x \,\|\, y$ we denote the concatenation of bit-strings $x, y \in \{0,1\}^*$. We write $\langle a, b, \ldots \rangle$ to denote an injective encoding of $a, b, \ldots$ into a bit-string that is uniquely decodable, where each of the encoded elements can have an arbitrary data type. If X is a finite set, we let $x \leftarrow\!\!\$\ X$ denote picking an element of X uniformly at random and assigning it to x. If mem is a table, $\mathsf{mem}[i]$ denotes the element of the table that is indexed by i. We let $\bot \notin \{0,1\}^*$ be an error code that indicates rejection. Explicitly uninitialized integers are assumed to be initialized to 0, Booleans to false, strings to $\bot$, and sets to the empty set. Uninitialized elements of a table are initialized similarly, based on the type of data they will hold.

Algorithms and Adversaries. By $y \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{O}}(x)$ we denote the randomized execution of algorithm $\mathcal{A}$ on input x, with oracle access to O, producing output y. We generalize this to multiple inputs or oracles in the natural way. If multiple oracles are provided, then $\mathcal{A}$ is allowed to call them arbitrarily many times, in any order. If $\mathcal{A}$ is deterministic, we use $\leftarrow$ instead. Running time is worst case. We let $[A(x)]$ denote the set of all possible outputs of A when invoked with input x. The instruction **abort**(x) is used to immediately halt an algorithm with output x. Adversaries are algorithms. We require that adversaries never pass $\bot$ as input to their oracles.

Security Games and Reductions. We use the code-based game-playing framework of [BR06] for specifying security definitions and writing proofs. We write "$\mathsf{require}\ d$" as shorthand for "$\mathtt{If}\ \neg d\ \mathtt{then\ return}\ \bot$". $\Pr[\mathcal{G}]$ denotes the probability that game $\mathcal{G}$ returns true. Variables in each game are shared with its oracles. In the security reductions, we omit specifying the running times of the constructed adversaries when they are roughly the same as the running time of the initial adversary. In our security-reduction games, we annotate some lines with comments of the form "$/\!/\ \mathcal{G}_{[i,j)}$" to indicate that a particular line belongs only to games $\mathcal{G}_i, \ldots, \mathcal{G}_{j-1}$; we write $j = \infty$ when the line belongs to all games through the end of the reduction. The lines not annotated with such comments are shared by all of the games that are shown in the particular figure. To determine the difference between $\mathcal{G}_{j-1}$ and $\mathcal{G}_j$, look for subscripts $[i, j)$ and $[j, k)$.

Fundamental Lemma of Game Playing. In our proofs, we frequently make use of the Fundamental Lemma of Game Playing [BR06]. Suppose that games $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ are syntactically identical except after a Boolean flag bad is set. Then

$$\Pr[\mathcal{G}_i] - \Pr[\mathcal{G}_{i+1}] \le \Pr[\mathsf{bad}^{\mathcal{G}_i}] = \Pr[\mathsf{bad}^{\mathcal{G}_{i+1}}],$$

where $\Pr[\mathsf{bad}^{\mathcal{G}}]$ denotes the probability of setting the flag bad in game $\mathcal{G}$.

2.1 Standard Primitives

We will use several standard cryptographic primitives including hash functions, function families, nonce-based encryption schemes, and digital signature schemes.

Hash Functions and Random Oracles. A hash function is a function $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^{\mathsf{H.ol}}$ for some fixed output length $\mathsf{H.ol} \in \mathbb{N}$. The advantage of an adversary $\mathcal{A}_{\mathsf{CR}}$ in breaking the collision resistance of H is defined as $\mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{CR}}) = \Pr[\,\mathsf{H}(x) = \mathsf{H}(y) \wedge x \neq y \ :\ (x, y) \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{CR}}\,]$. Our use of keyless hash functions follows the human ignorance interpretation of Rogaway [Rog06]. Some of our analysis will be performed in the random oracle model in which a function is chosen at random.

Nonce-Based Encryption. A nonce-based encryption scheme NE specifies deterministic algorithms $\mathsf{NE.Enc}$ and $\mathsf{NE.Dec}$. Associated to every instance of NE is a key length $\mathsf{NE.kl} \in \mathbb{N}$, a nonce length $\mathsf{NE.nl} \in \mathbb{N}$, and an associated-data space $\mathsf{NE.AD}$. The encryption algorithm $\mathsf{NE.Enc}$ takes a symmetric key $K \in \{0,1\}^{\mathsf{NE.kl}}$, a nonce $n \in \{0,1\}^{\mathsf{NE.nl}}$, a plaintext $m \in \{0,1\}^*$, and associated data $ad \in$

$\mathsf{NE.AD}$ to return a ciphertext c. The decryption algorithm $\mathsf{NE.Dec}$ takes K, n, c, ad to return a plaintext $m \in \{0,1\}^* \cup \{\perp\}$, where $\perp$ indicates a failure to recover a plaintext. Decryption correctness requires that for all $K \in \{0,1\}^{\mathsf{NE.kl}}$, $n \in \{0,1\}^{\mathsf{NE.nl}}$, $m \in \{0,1\}^*$, and $ad \in \mathsf{NE.AD}$, the following holds: $\mathsf{NE.Dec}(K, n, \mathsf{NE.Enc}(K, n, m, ad), ad) = m$. The tidiness property [NRS14] requires that $\mathsf{NE.Enc}(K, n, \mathsf{NE.Dec}(K, n, c, ad), ad) = c$ whenever $\mathsf{NE.Dec}(K, n, c, ad) \neq \perp$. It implies that $\mathsf{NE.Dec}(K, n, c, ad) = \mathsf{NE.Dec}(K, n, c', ad) \neq \perp$ is impossible for $c \neq c'$.

Associated Data in Nonce-Based Encryption. We define every nonce-based encryption scheme NE with respect to an associated-data space $\mathsf{NE.AD}$ that contains either all bit-strings (i.e. $\mathsf{NE.AD} = \{0,1\}^*$) or a single element (i.e. $\mathsf{NE.AD} = \{\varepsilon\}$). For the sake of simplicity, in the latter case, we omit the associated data field ad from the syntax of all NE algorithms.

Digital Signatures. A digital signature scheme DS specifies algorithms $\mathsf{DS.Kg}$, $\mathsf{DS.Sig}$ and $\mathsf{DS.Ver}$, where $\mathsf{DS.Ver}$ is deterministic. Associated to every instance of DS is a signing key length $\mathsf{DS.skl} \in \mathbb{N}$ and a signature length $\mathsf{DS.sl} \in \mathbb{N}$. The key generation algorithm $\mathsf{DS.Kg}$ returns a key pair (sk, vk) where $sk \in \{0,1\}^{\mathsf{DS.skl}}$ is a signing key and vk is the corresponding verification key. The signing algorithm $\mathsf{DS.Sig}$ takes sk and a message $m \in \{0,1\}^*$ to return a signature $s \in \{0,1\}^{\mathsf{DS.sl}}$. The deterministic verification algorithm $\mathsf{DS.Ver}$ takes vk, m, s to return a decision $d \in \{\mathsf{true}, \mathsf{false}\}$ regarding whether s is a valid signature of m under vk. Correctness condition requires that for all $(sk, vk) \in [\mathsf{DS.Kg}]$, all $m \in \{0,1\}^*$, and all $s \in [\mathsf{DS.Sig}(sk, m)]$, the following holds: $\mathsf{DS.Ver}(vk, m, s) = \mathsf{true}$.

2.2 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 we nonetheless know that 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 normally 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 [BS20], and will likewise be needed in this work.

Our security model for symmetric signcryption in Section 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 adversary $\mathcal{A}_{\mathsf{AEAD}}$ 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 $\textsc{Enc}$, and decryption oracle $\textsc{Dec}$. Among the group oracles, $\textsc{NewHonGroup}$ creates new groups with honestly generated NE keys, $\textsc{ExposeGroup}$ reveals the keys of existing groups, and $\textsc{NewCorrGroup}$ instantiates new corrupt groups with NE keys of $\mathcal{A}_{\mathsf{AEAD}}$'s choice. The encryption oracle $\textsc{Enc}$ takes a group identifier g, a nonce n, two challenge messages m_0, m_1, and associated data ad; it returns the encryption of m_b under the provided parameters. We require NE to be nonce-misuse resistant [RS06], 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. To prevent trivial wins, 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 [Shr04,RS06], it returns $\bot$ if $b = 0$, and it returns m otherwise. To prevent trivial wins, 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 $\mathcal{G}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}})$
$b \leftarrow\!\$ \{0,1\}$ ; $b' \leftarrow\!\$ \mathcal{A}^{\mathrm{G},\textsc{Enc},\textsc{Dec}}_{\mathsf{AEAD}}$ ; Return $b = b'$

$\textsc{Enc}(g, n, m_0, m_1, ad)$
require $\mathsf{K}[g] \neq \bot$ 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 $\bot$
    $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_b, ad)$
$N_0 \leftarrow N_0 \cup \{(g, n, m_0, ad)\}$
$N_1 \leftarrow N_1 \cup \{(g, n, m_1, ad)\}$
$C \leftarrow C \cup \{(g, n, c, ad)\}$ ; Return $c$

$\textsc{Dec}(g, n, c, ad)$
require $\mathsf{K}[g] \neq \bot$ 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 $\bot$ else return $m$
```

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

$\textsc{Enc}(g, n, m, ad)$
require $\mathsf{K}[g] \neq \bot$
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 $c$

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

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

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

$\textsc{ExposeGroup}(g)$
require $\mathsf{K}[g] \neq \bot$ $\boxed{\text{and } \neg\mathsf{chal}[g]}$
$\mathsf{group\_is\_corrupt}[g] \leftarrow \mathsf{true}$
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. 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 $\mathrm{G} = \{\textsc{NewHonGroup}, \textsc{ExposeGroup}, \textsc{NewCorrGroup}\}$ that are provided to an adversary in either game, except that the $\boxed{\text{boxed}}$ code only appears in the 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 $\mathsf{Adv}^{\mathsf{KR}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{KR}}) = \Pr[\mathcal{G}^{\mathsf{KR}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{KR}})]$. Adversary $\mathcal{A}_{\mathsf{KR}}$ is given group oracles G, encryption oracle Enc, decryption oracle Dec, and key-guessing oracle Guess. The group oracles G match those from $\mathcal{G}^{\mathsf{AEAD}}$. The encryption oracle Enc takes a group identifier g, a nonce n, a message m, and associated data ad, and returns the encryption of m under the provided parameters. For consistency with other games, we disallow querying a message m across two distinct calls to Enc with repeated g, n, and ad values. The decryption oracle Dec takes g, n, c, ad as input and returns the decryption m of c under the corresponding parameters. The Guess oracle takes key K as input and sets the win flag if K is an honest group key. Note that KR security of NE is implied by its AEAD security. Meaning that for any adversary $\mathcal{A}_{\mathsf{KR}}$ against the KR security of NE, one can build an adversary $\mathcal{A}_{\mathsf{AEAD}}$ against the AEAD security of NE such that $\mathcal{A}_{\mathsf{AEAD}}$ wins whenever $\mathcal{A}_{\mathsf{KR}}$ does. Intuitively, if $\mathcal{A}_{\mathsf{KR}}$ successfully guesses an honest key then $\mathcal{A}_{\mathsf{AEAD}}$ can use that key to decrypt challenge ciphertexts previously returned by its Enc oracle. The decryption correctness of NE guarantees that $\mathcal{A}_{\mathsf{AEAD}}$ would be able to deduce the challenge bit.

```
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 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 digital 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 $\mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}}) = \Pr[\mathcal{G}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})]$. Adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ is given access to user oracles U, where oracle NewHonUser creates new users with an honestly generated DS key pairs, oracle ExposeUser reveals signing keys of existing users, and oracle NewCorrUser instantiates new corrupt users with DS key pairs of $\mathcal{A}_{\mathsf{SUFCMA}}$'s choice. Adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ is also given signing oracle Sign that on input (u, m) signs message m with the signing key of user u. The game requires $\mathcal{A}_{\mathsf{SUFCMA}}$ to forge a signature s for any user-message pair (u, m) with honest user u such that s was not previously returned in response to a $\textsc{Sign}(u, m)$ query.

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.

```
$(sk, vk) \leftarrow\$ \mathsf{SS.UserKg}$
$c \leftarrow\$ \mathsf{SS.SigEnc}(g, K_g, u, sk_u, n, m, ad)$
$m \leftarrow \mathsf{SS.VerDec}(g, K_g, u, vk_u, n, c, ad)$
```

Fig. 6: Syntax of a symmetric signcryption scheme SS.

Syntax. A symmetric signcryption scheme SS specifies algorithms $\mathsf{SS.UserKg}$, $\mathsf{SS.SigEnc}$, $\mathsf{SS.VerDec}$, where $\mathsf{SS.VerDec}$ is deterministic. The syntax used for the algorithms of SS is given in Fig. 6. 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 $m \in \{0,1\}^* \cup \{\bot\}$, where $\bot$ indicates a failure to recover a plaintext. We say that SS is *deterministic* $\mathsf{SS.SigEnc}$ is deterministic.

Correctness. The decryption correctness of a symmetric signcryption scheme SS requires that for all $g \in \{0,1\}^*$, all $K_g \in \{0,1\}^{\mathsf{SS.gkl}}$, all $u \in \{0,1\}^*$, all $(sk_u, vk_u) \in [\mathsf{SS.UserKg}]$, all $n \in \mathsf{SS.NS}$, all $m \in \mathsf{SS.MS}$, all $ad \in \mathsf{SS.AD}$, and all $c \in [\mathsf{SS.SigEnc}(g, K_g, u, sk_u, n, m, ad)]$ the following holds: $\mathsf{SS.VerDec}(g, K_g, u, vk_u, n, c, ad) = m$

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 SealPacket encryption algorithm in Keybase (as defined in Section 5) 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 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: The Basics. Consider game $\mathcal{G}^{\mathsf{IUF}}$ of Fig. 7, 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.

IUF Game: User and Group Oracles. Among user oracles, NewHonUser creates honest users with honestly generated signing keys, NewCorrUser creates corrupt users with malicious signing keys, and ExposeUser exposes the 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 user_is_corrupt and group_is_corrupt to keep track of the users and groups whose keys are not secure, respectively. The IUF game never checks group_is_corrupt, deliberately giving the adversary full control over group keys.

IUF Game: The SigEnc Oracle. 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 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.

IUF Game: The VerDec Oracle. 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 SS.VerDec not directly chosen by the adversary. If $m \neq \bot$, 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 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 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. 8 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

```
Game $\mathcal{G}^{\mathsf{IUF}}_{\mathsf{SS},\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}})$
    $\mathcal{A}_{\mathsf{IUF}}^{\mathrm{U},\mathrm{G},\textsc{SigEnc},\textsc{VerDec}}$
    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 $\mathcal{G}^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{OAE}})$
    $b \leftarrow\!\!\$\ \{0,1\}$ ; $b' \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{OAE}}^{\mathrm{U},\mathrm{G},\textsc{SigEnc},\textsc{VerDec}}$
    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. 7: **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.

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

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

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

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

Fig. 8: Sample ciphertext-triviality predicates which capture rules for deciding if a successfully decrypted VerDec query was trivially obtainable or forgeable.

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 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 SealPacket encryption algorithm (as defined in Section 5) 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: The Basics. Consider game $\mathcal{G}^{\mathsf{OAE}}$ of Fig. 7 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 [Shr04,Rog04], where an adversary can learn the challenge bit by forging a ciphertext to its decryption oracle.

OAE Game: User and Group Oracles. 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 user_is_corrupt, deliberately giving the adversary full control over user keys.

OAE Game: The SigEnc Oracle. 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 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 [RS06], 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.

OAE Game: The VerDec Oracle. 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-$\perp$ 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 $\perp$ 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 $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{sec}}$ 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. 9: 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. 9 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, $\mathsf{pred}_{\mathsf{trivial}}(z, \{((g, u, n, m^*, ad), c)\})$ 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

```
StE.UserKg
    $(sk, vk) \leftarrow\$ \mathsf{DS.Kg}$ ; Return $(sk, vk)$

StE.SigEnc$(g, K_g, u, sk_u, n, m, ad)$
    $m_s \leftarrow \langle g, n, m, ad \rangle$
    $s \leftarrow\$ \mathsf{DS.Sig}(sk_u, m_s)$
    $m_e \leftarrow s \,\|\, m$ ; $ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$
    $c \leftarrow \mathsf{NE.Enc}(K_g, n, m_e, ad_{\mathsf{NE}})$ ; Return $c$

StE.VerDec$(g, K_g, u, vk_u, n, c, ad)$
    $ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$
    $m_e \leftarrow \mathsf{NE.Dec}(K_g, n, c, ad_{\mathsf{NE}})$
    If $m_e = \bot$ then return $\bot$
    $s \,\|\, m \leftarrow m_e$    // s.t. $|s| = \mathsf{DS.sl}$, $|m| \geq 0$
    $m_s \leftarrow \langle g, n, m, ad \rangle$
    If $\neg \mathsf{DS.Ver}(vk_u, m_s, s)$ then return $\bot$
    Return $m$
```

Fig. 10: Signcryption scheme $\mathsf{StE} = \mathsf{SIGN\text{-}THEN\text{-}ENCRYPT\text{-}SS}[\mathsf{DS}, \mathsf{NE}]$.

$\mathsf{pred}_{\mathsf{trivial}}(z, C)$ for SS, despite not knowing the challenge bit b that is needed to explicitly build C. This is true in all of our proofs.)

THE NECESSITY OF C. It would not suffice to use Q instead of C for identifying trivial forgeries. Consider an SS that produces malleable ciphertexts which we want to analyze with respect to the existential unforgeability predicate $\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}$. Let Q contain an entry that maps one of m_0, m_1 to c. Then upon VERDEC decrypting some $c' \neq c$ into m_0, the set Q does not allow to distinguish between the following two cases: (1) if c is an encryption of m_0 then c' could have been trivially obtained by mauling c; (2) if c is an encryption of m_1 then c' might represent an actual forgery that was not obtained in a trivial way.

4 Symmetric Signcryption from Encryption and Signatures

In this section, we build symmetric signcryption schemes by sequentially composing a digital signature scheme with a nonce-based encryption scheme. We prove that one of constructions achieves strong variants of the security notions that we defined in Section 3. In particular, we compose an SUFCMA-secure digital signature scheme with an AEAD-secure nonce-based encryption scheme using the Sign-then-Encrypt (StE) and Encrypt-then-Sign (EtS) compositions. The StE-based symmetric signcryption scheme achieves IUF security with ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ and achieves OAE security with $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ and output-guarding function $\mathsf{func}^{\mathsf{silence\text{-}with\text{-}m_1}}_{\mathsf{out}}[\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}]$. The straightforward proofs for these results are in Appendix A. Note that the same result can also be proved for the IUF security of the EtS-based scheme. However, proving the same about the OAE security of the EtS-based scheme would come at the cost of introducing the non-standard assumption that the underlying digital signature scheme provides unique signatures. We explain the need for this assumption below.

4.1 Sign-then-Encrypt Based Construction

Our StE-based symmetric signcryption scheme StE is shown in Fig. 10. Of it, we prove the following results.

Construction 1. *Let* DS *be a digital signature scheme. Let* NE *be a nonce-based encryption scheme with associated-data space* $\mathsf{NE.AD} = \{0,1\}^*$. *Then* $\mathsf{StE} = \mathsf{SIGN\text{-}THEN\text{-}ENCRYPT\text{-}SS}[\mathsf{DS}, \mathsf{NE}]$ *is the symmetric signcryption scheme as defined in Fig. 10, with group-key length* $\mathsf{StE.gkl} = \mathsf{NE.kl}$, *nonce space* $\mathsf{StE.NS} = \{0,1\}^{\mathsf{NE.nl}}$, *message space* $\mathsf{StE.MS} = \{0,1\}^*$, *and associated-data space* $\mathsf{StE.AD} = \{0,1\}^*$.

Theorem 1. *Let* $\mathsf{StE} = \mathsf{SIGN\text{-}THEN\text{-}ENCRYPT\text{-}SS}[\mathsf{DS}, \mathsf{NE}]$ *be the symmetric signcryption scheme built from some* DS, NE *as specified in Construction 1. Let* $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 8. Let* $\mathcal{A}_{\mathsf{IUF}}$ *be any adversary against the* IUF *security of* StE *with respect to* $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$. *Then we can build an adversary* $\mathcal{A}_{\mathsf{SUFCMA}}$ *against the* SUFCMA *security of* DS *such that*

$$\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{StE}, \mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}}).$$

Let $\mathsf{func}_{\mathsf{out}}^{\perp}$ *be the output-guarding function as defined in Fig. 9. Let* $\mathcal{A}_{\mathsf{OAE}}$ *be any adversary against the* OAE *security of* StE *with respect to* $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$ *and* $\mathsf{func}_{\mathsf{out}}^{\perp}$*. Then we can build adversary* $\mathcal{A}_{\mathsf{AEAD}}$ *against the* AEAD *security of* NE *such that*

$$\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{StE},\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}},\mathsf{func}_{\mathsf{out}}^{\perp}}(\mathcal{A}_{\mathsf{OAE}}) \le \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}}).$$

4.2 Encrypt-then-Sign Based Construction

We now present the formal definition of our EtS construction. We omit proving the security of this construction. The IUF security proof for EtS would follow similar steps as the one for StE. The OAE security proof would require us to introduce an additional assumption about the signature scheme.

```
EtS.UserKg
  $(sk, vk) \leftarrow\$ \mathsf{DS.Kg}$ ; Return $(sk, vk)$

EtS.SigEnc($g, K_g, u, sk_u, n, m, ad$)
  $ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$
  $c_{\mathsf{NE}} \leftarrow \mathsf{NE.Enc}(K_g, n, m, ad_{\mathsf{NE}})$
  $m_s \leftarrow \langle g, n, c_{\mathsf{NE}}, ad \rangle$
  $s \leftarrow\$ \mathsf{DS.Sig}(sk_u, m_s)$ ; Return $(c_{\mathsf{NE}}, s)$

EtS.VerDec($g, K_g, u, vk_u, n, c, ad$)
  $(c_{\mathsf{NE}}, s) \leftarrow c$
  $m_s \leftarrow \langle g, n, c_{\mathsf{NE}}, ad \rangle$
  If $\neg \mathsf{DS.Ver}(vk_u, m_s, s)$ then return $\perp$
  $ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$
  $m \leftarrow \mathsf{NE.Dec}(K_g, n, c_{\mathsf{NE}}, ad_{\mathsf{NE}})$
  Return $m$
```

Fig. 11: Symmetric signcryption scheme EtS = ENCRYPT-THEN-SIGN-SS[NE, DS].

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

OAE Security of EtS Requires Unique Signatures. At the beginning of this section, we claimed that proving the OAE security of EtS with respect to $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$ and $\mathsf{func}_{\mathsf{out}}^{\mathsf{silence\text{-}with\text{-}m_1}}[\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}]$ would require the assumption that the signature scheme produces unique signatures. Here we provide the intuition for why this assumption is needed. Consider a digital signature scheme that does not produce unique signatures. Then the adversary could expose an honest user in the OAE security game (or create a corrupt user with the same signing key as that of another honest user), and use their signing key to produce a new signature over an existing, honestly generated ciphertext c_{NE} (i.e. one that was produced as a result of a prior SigEnc query). It could then query the ciphertext c_{NE} along with the new signature to the VerDec oracle to trivially win the game. This attack can only be mitigated if DS produces unique signatures. As digital signature schemes standardized in NIST's FIPS 186-5 standard [CMRR23] (including the RSA DSA, ECDSA, and EDDSA) do not guarantee unique signatures, we do not analyze the OAE security of the EtS construction in this work.

5 Modeling 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. 12 and visual depictions of their data flows in Fig. 13. We omit the decryption algorithms BoxMessage.VerDec and SealPacket.VerDec from Fig. 12 as Keybase's implementation of these algorithms follows naturally from the corresponding

```
$\underline{\mathsf{BoxMessage.SigEnc}(g, K_g, u, sk_u, n, m_{\mathsf{body}}, ad)}$   $//$ 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. 12: 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.

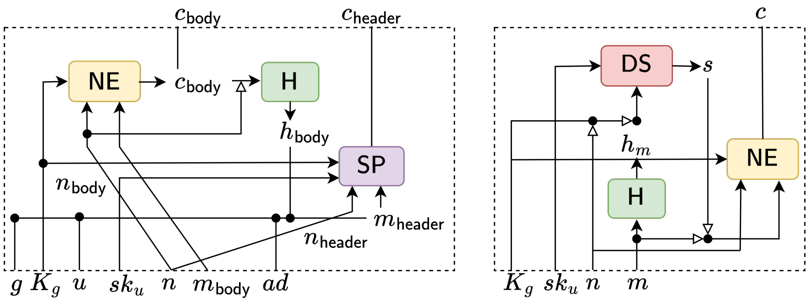

Fig. 13: The BoxMessage/BM (**Left**) and SealPacket/SP (**Right**) algorithms used to encrypt chat messages and message headers in Keybase. We omit the inputs g, u, and ad from SP as it does not use g or u and $ad = \varepsilon$ always holds.

SigEnc algorithms. We define the VerDec algorithms explicitly in our formalization 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 Section 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 set $\mathsf{SealPacket.AD} = \{\varepsilon\}$, 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 Appendix E 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 [BRS03]. 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. 12 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.

We highlight the consequences of repetition across either of n_{body} or n_{header}. The XSalsa20-Poly1305 authenticated encryption scheme combines the XSalsa20 stream cipher and the Poly1305 *one-time* message authentication code. The XSalsa20 stream cipher XORs plaintexts with a keystream that is deterministically derived from its key and nonce. So nonce repetition allows learning the XOR of the underlying messages. This is not an issue for twice repetition of n_{header} the header plaintext m_{header} is sent in the clear with ciphertexts in Keybase (a detail omitted from our abstraction), but reveals confidential information if the repetition involves a n_{body}. Moreover, the Poly1305 key is extracted from the first 32 keystream bytes. Nonce repetition results in key reuse, allowing an attacker to create forgeries for this key.

5.1 Message and Header Encryption Schemes BoxMessage and SealPacket

MESSAGE ENCRYPTION SCHEME BM. Our modular symmetric signcryption construction BM models the BoxMessage chat-encryption algorithm as follows.

```
$\underline{\mathsf{BM.UserKg}}$
$(sk, vk) \leftarrow\!\!\$\ \mathsf{SP.UserKg}$ ; 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. 14: 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. 15: 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.

Construction 3. *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. 14, 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 $\mathsf{BM.NS} = \{0,1\}^{\mathsf{NE.nl}} \times \mathsf{SP.NS}$.*

HEADER ENCRYPTION SCHEME SP. Our modular symmetric signcryption construction SP models the header-encryption algorithm $\mathsf{SealPacket}$ as follows.

Construction 4. *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 deterministic symmetric signcryption scheme as defined in Fig. 15, 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* $\mathsf{SP.AD} = \{\varepsilon\}$*.*

6 Security Analysis of Keybase Chat Encryption

In this section we analyze the security of the symmetric signcryption schemes BM and SP defined in Section 5. In Section 6.1, we show the in-group unforgeability of BM and SP. In Sections 6.2 and 6.3, 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. 7. 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.

6.1 In-Group Unforgeability of BoxMessage and SealPacket

In-Group Unforgeability of BoxMessage. We reduce the in-group unforgeability of the symmetric signcryption scheme $\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}]$ to the security of its underlying primitives. In particular, we reduce the IUF security of $\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}]$ to the IUF security of SP and the collision resistance of 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.

Theorem 2. *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 3. 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. 8. 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*

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

The formal proof of Theorem 2 is in Appendix B.1.

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 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.

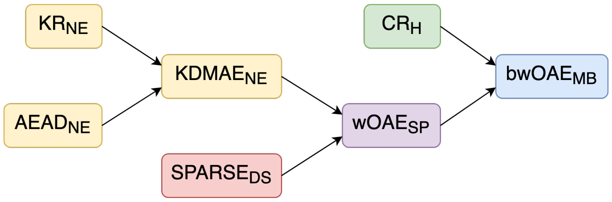
$\mathsf{KR}_{\mathsf{NE}}$
$\mathsf{AEAD}_{\mathsf{NE}}$
$\mathsf{KDMAE}_{\mathsf{NE}}$
$\mathsf{SPARSE}_{\mathsf{DS}}$
CR_{H}
$\mathsf{wOAE}_{\mathsf{SP}}$
$\mathsf{bwOAE}_{\mathsf{MB}}$

Fig. 16: A summary of the reductions that we provide for the wOAE security of SP and the bwOAE security of BM.

Theorem 3. *Let* $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{H}, \mathsf{DS}, \mathsf{NE}]$ *be the symmetric signcryption scheme built from some* H, DS, *and* NE *as specified in Construction 4. Let* $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 8. 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*

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

The formal proof of Theorem 3 is in Appendix B.2.

6.2 Out-Group AE Security of BoxMessage

Out-group AE security of $\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}]$ reduces to the security of its underlying primitives as summarized by the rightmost arrows of Fig. 16. 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 built from the authenticated encryption scheme XSalsa20-Poly1305 that is 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. 7 (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*

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

In addition, the instructions updating the nonce set

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

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. 7 (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*

$$\textsf{require } \forall d, (g,u,n,m_d,ad) \notin N_d \quad \textit{is replaced with} \quad \begin{array}{l} (n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n \\ \textit{If } n_{\mathsf{body}} = n_{\mathsf{header}} \textit{ then return } \bot \\ \textit{If } (g, n_{\mathsf{body}}) \in N \textit{ then return } \bot \\ \textit{If } (g, n_{\mathsf{header}}) \in N \textit{ then return } \bot. \end{array}$$

In addition, the instructions updating the nonce set

$$\begin{array}{l} N_0 \leftarrow N_0 \cup \{(g,u,n,m_0,ad)\} \\ N_1 \leftarrow N_1 \cup \{(g,u,n,m_1,ad)\} \end{array} \quad \textit{are replaced with} \quad \begin{array}{l} N \leftarrow N \cup \{(g,n_{\mathsf{header}})\} \\ N \leftarrow N \cup \{(g,n_{\mathsf{body}})\}. \end{array}$$

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 $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$. 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 $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ captures this by providing the adversary access to an NE encryption oracle $\textsc{Enc}$ in addition to the $\textsc{SigEnc}$ and $\textsc{VerDec}$ 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 $\textsc{SigEnc}$ 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 Section 5 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.*

```
$\underline{\textsc{Enc}[\mathcal{M}, \mathsf{NE}](g, n_{\mathsf{body}}, m_{\mathsf{body},0}, m_{\mathsf{body},1})}$
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, $\mathsf{group\_is\_corrupt}$, chal with the rest of the security game. We denote the resulting game (and security notion) by $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$. 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 $\textsc{Enc}$ 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 $\mathsf{BoxMessage}$. We prove bwOAE security of BM.

Theorem 4. *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 3. 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. 8. Let $\mathsf{func}^{\bot}_{\mathsf{out}}$ be the output-guarding functions as defined in Fig. 9. 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*

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

The formal proof of Theorem 4 is in Appendix C.4.

Note that we prove the OAE security of BM with respect to $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ and $\mathsf{func}^{\bot}_{\mathsf{out}}$. As discussed in Section 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 $\mathsf{func}^{\bot}_{\mathsf{out}}$. 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}.

6.3 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. 16). In particular, $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ security holds (for any $\mathcal{M} \subseteq \{0,1\}^*$) 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 $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ security game, the adversary is provided with signcryption and unsigncryption oracles $\textsc{SigEnc}$ and $\textsc{VerDec}$ for SP, and an encryption oracle $\textsc{Enc}$ 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 $\textsc{VerDec}$; (b) make left-or-right queries to its NE encryption oracle $\textsc{Enc}$; (c) make left-or-right queries to its SP encryption oracle $\textsc{SigEnc}$.

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 $\textsc{Enc}$ 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.[3]

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 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.

The Strength of the Targeted Security Notion. During the analysis of the out-group AE security of BM in Section 6.2, we discussed that the left-or-right security of SP is not required for us to prove the security of BM. Meaning that the requirement (c) listed above arose as a result of us choosing to define the $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ game in a way that it captures a stronger security notion than strictly necessary. We note that in our $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ security analysis of SP, case (c) follows from case (b), and does not need additional assumptions.

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. 17, 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 can 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.

We verify our intuition about the $\mathcal{M}$-sparsity of the $\mathsf{Ed25519}$ signature scheme underlying SP in Appendix C.1. $\mathsf{Ed25519}$ is a deterministic signature scheme introduced by Bernstein, Duif, Lange, Schwabe, and Yang in [BDL+11]. 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 can 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.

The Message Space $\mathcal{M}$. Keybase uses the MessagePack serialization format [Fur] 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

[3] 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.

```
Game $\mathcal{G}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi}(\mathcal{A}_{\mathsf{KDMAE}})$
$b \leftarrow\!\$\ \{0,1\}$ ; $b' \leftarrow\!\$\ \mathcal{A}^{\mathrm{G,Enc,Dec}}_{\mathsf{KDMAE}}$
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. 18: Game defining authenticated-encryption security of NE for Φ-key-dependent messages, where Φ is a class of message-deriving functions and $\mathrm{G} = \{\textsc{NewHonGroup}, \textsc{ExposeGroup}, \textsc{NewCorrGroup}\}$.

about the data structure happens to 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) Φ of message-deriving functions. For simplicity, 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 $\|\phi\|$.

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

Fig. 17: Game defining $\mathcal{M}$-sparsity of a digital signature scheme DS for a set $\mathcal{M}$.

AE Security of NE for Key-Dependent Messages. Consider game $\mathcal{G}^{\mathsf{KDMAE}}$ of Fig. 18, defined for a nonce-based encryption scheme NE, a class of message-deriving functions Φ, and an adversary $\mathcal{A}_{\mathsf{KDMAE}}$. The advantage of $\mathcal{A}_{\mathsf{KDMAE}}$ in breaking the Φ-KDMAE security of NE is defined as $\mathsf{Adv}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi}(\mathcal{A}) = 2 \cdot \Pr[\mathcal{G}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi}(\mathcal{A})] - 1$. 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 $\phi_0, \phi_1 \in \Phi$ as input (rather than challenge messages m_0, m_1). 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 [BRS03,BPS07,BK11,BMT14]. There are strong impossibility results [BK11] regarding 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}. At the beginning of this section we discussed that in the $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ security game for SP, the $\textsc{SigEnc}$ and $\textsc{Enc}$ oracles can be thought of as returning an NE ciphertext that encrypts an output of some message-deriving function. For example, the $\textsc{Enc}$ oracle can be expressed roughly as $\textsc{Enc}(g, n, m_0, m_1) := \mathsf{NE.Enc}(\mathsf{K}[g], n, \phi_{m_b}(\mathsf{K}[g]))$ with respect to any class Φ that for every $m \in \{0,1\}^*$ contains the constant function $\phi_m(K) := m$ (i.e. ϕ_m ignores its input K). We now define the class Φ_{SP} containing all message-deriving functions that are used by either SigEnc or Enc.

Construction 5. *Let* NE *be a nonce-based encryption scheme. Let* H *be a hash function. Let* DS *be a digital signature scheme. Let* $\mathsf{SIGENC\text{-}DER}$ *and* $\mathsf{ENC\text{-}DER}$ *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 message-deriving functions that is defined as follows.*

$$\Phi_{\mathsf{SP}} = \left\{ \mathsf{SIGENC\text{-}DER}[\mathsf{NE}, \mathsf{H}, \mathsf{DS}, m, n, sk] \;\middle|\; m \in \{0,1\}^*,\ n \in \{0,1\}^{\mathsf{NE.nl}},\ sk \in \{0,1\}^{\mathsf{DS.skl}} \right\} \\ \cup \left\{ \mathsf{ENC\text{-}DER}[m] \;\middle|\; m \in \{0,1\}^* \right\}.$$

Note that $\mathsf{SIGENC\text{-}DER}$ only uses K as a part of the message m_s signed by $\mathsf{DS.Sig}$. Keybase instantiates DS with $\mathsf{Ed25519}$ which computes two $\mathsf{SHA\text{-}512}$ 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 [BDD23] (for $\mathsf{SHA\text{-}512}$ with output reduced modulo a prime) we capture $\mathsf{SIGENC\text{-}DER}$ 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* $\Phi = \left\{ \phi_\gamma \;\middle|\; \phi_\gamma(\cdot) = \gamma(\mathsf{H}(\cdot)), \gamma \in \Gamma \right\}$.

In Appendix C.2 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* 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 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 $\mathcal{A}_{\mathsf{KDMAE}}$ 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 Appendix C.3.

Out-group AE Security of $\mathsf{SealPacket}$. We prove wOAE security of SP.

Theorem 5. *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 4. Let* $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 8. Let* $\mathsf{func}^{\perp}_{\mathsf{out}}$ *be the output-guarding function as defined in Fig. 9. 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 5. 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 formal proof of Theorem 5 is in Appendix C.5.

We prove $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M},\mathsf{NE}]]$ security of SP with the ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ and the output-guarding function $\mathsf{func}^{\perp}_{\mathsf{out}}$. The predicate $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ requires ciphertext integrity, except ciphertexts do not bind to the user's identifier. As pointed out in Section 3 and Section 6.2, the choice of this predicate follows from the level of ciphertext integrity achieved by SP. The use of $\mathsf{func}^{\perp}_{\mathsf{out}}$ makes the $\textsc{VerDec}$ oracle less informative. We chose $\mathsf{func}^{\perp}_{\mathsf{out}}$ despite that fact because proving security with respect to a stronger parameter like $\mathsf{func}^{\mathsf{silence\text{-}with\text{-}m_1}}_{\mathsf{out}}[\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}]$ comes at the cost of adding complexity to our reductions.

6.4 Stronger Out-Group AE Security Results

The OAE security results that we showed in Sections 6.2 and 6.3 used the output guarding function $\mathsf{func}^{\perp}_{\mathsf{out}}$. In this section, we show that for the symmetric signcryption schemes that we consider in this work, the OAE security of a scheme with respect to $\mathsf{func}^{\perp}_{\mathsf{out}}$ implies its OAE security with respect to the stronger output guarding function $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$. This requires us to first introduce a new security notion for digital signature schemes and symmetric signcryption schemes that we call unique verifiability.

Unique Verifiability. Consider game $\mathcal{G}^{\mathsf{UV}}$ of Fig. 19, defined for a digital signature scheme DS and an adversary $\mathcal{A}_{\mathsf{UV}}$ on the left, and for a symmetric signcryption scheme SS and an adversary $\mathcal{A}_{\mathsf{UV}}$ on the right. The advantage of $\mathcal{A}_{\mathsf{UV}}$ in breaking the UV security of DS (resp. SS) is defined as $\mathsf{Adv}^{\mathsf{UV}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{UV}}) = \Pr[\mathcal{G}^{\mathsf{UV}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{UV}})]$ (resp. $\mathsf{Adv}^{\mathsf{UV}}_{\mathsf{SS}}(\mathcal{A}_{uv}) = \Pr[\mathcal{G}^{\mathsf{UV}}_{\mathsf{SS}}(\mathcal{A}_{\mathsf{UV}})]$).

Intuitively, for DS, this game captures the inability of an attacker to produce a signature by executing the underlying signing algorithm with some signing key sk such that the resulting signature verifies successfully under an attacker-chosen verification key vk' that is different from the "honest" verification key corresponding to sk. Analogously, for SS, the game captures the inability of an attacker to produce a ciphertext by executing the underlying encryption algorithm with some signing key sk such that the resulting ciphertext decrypts successfully under an attacker-chosen verification key vk' that is different from the "honest" verification key corresponding to sk.

```
$\mathcal{G}^{\mathsf{UV}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{UV}})$
$(sk, vk', m) \leftarrow\!\$\ \mathcal{A}_{\mathsf{UV}}$
$s \leftarrow\!\$\ \mathsf{DS.Sig}(sk, m)$
$\mathsf{win}_0 \leftarrow \mathsf{DS.Ver}(vk', m, s)$
$\mathsf{win}_1 \leftarrow ((sk, vk') \notin [\mathsf{DS.Kg}])$
Return $\mathsf{win}_0$ and $\mathsf{win}_1$
```

```
$\mathcal{G}^{\mathsf{UV}}_{\mathsf{SS}}(\mathcal{A}_{\mathsf{UV}})$
$(g, K_g, u, u', sk_u, vk_{u'}, n, m, ad) \leftarrow\!\$\ \mathcal{A}_{\mathsf{UV}}$
$c \leftarrow\!\$\ \mathsf{SS.SigEnc}(g, K_g, u, sk_u, n, m, ad)$
$\mathsf{win}_0 \leftarrow (\mathsf{SS.VerDec}(g, K_g, u', vk_{u'}, n, c, ad) \neq \perp)$
$\mathsf{win}_1 \leftarrow ((sk_u, vk_{u'}) \notin [\mathsf{SS.UserKg}])$
Return $\mathsf{win}_0$ and $\mathsf{win}_1$
```

Fig. 19: **Left pane:** Game defining the unique verifiability of a digital signature scheme DS. **Right pane:** Game defining the unique verifiability of a symmetric signcryption scheme DS.

As an intermediate result, we first show that the unique verifiability of the symmetric signcryption schemes $\mathsf{StE} = \mathsf{SIGN\text{-}THEN\text{-}ENCRYPT\text{-}SS}[\mathsf{DS},\mathsf{NE}]$, $\mathsf{EtS} = \mathsf{ENCRYPT\text{-}THEN\text{-}SIGN\text{-}SS}[\mathsf{NE},\mathsf{DS}]$, and $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{H},\mathsf{DS},\mathsf{NE}]$ defined in Sections 4 and 5 is implied by the unique verifiability of the underlying digital signature scheme DS. In Appendix D.2 we prove that in the random oracle model, the $\mathsf{Ed25519}$ digital signature scheme underlying SP achieves unique verifiability. We omit

$\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}]$ from Proposition 2 for simplicity, but it is easy to see that the unique verifiability of BM follows from the unique verifiability of SP.

Proposition 2. *Let* $\mathsf{StE} = \mathsf{SIGN\text{-}THEN\text{-}ENCRYPT\text{-}SS}[\mathsf{DS}, \mathsf{NE}]$, $\mathsf{EtS} = \mathsf{ENCRYPT\text{-}THEN\text{-}SIGN\text{-}SS}[\mathsf{NE}, \mathsf{DS}]$, *and* $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{H}, \mathsf{DS}, \mathsf{NE}]$ *be symmetric signcryption schemes built from some* $\mathsf{DS}, \mathsf{NE}, \mathsf{H}$ *as specified in Construction 1, Construction 2, and Construction 4 respectively. Let* $\mathsf{SS} \in \{\mathsf{StE}, \mathsf{EtS}, \mathsf{SP}\}$*. Let* $\mathcal{A}^{\mathsf{SS}}_{\mathsf{UV}}$ *be any adversary against the* UV *security of* SS*. Then we can build adversary* $\mathcal{A}^{\mathsf{DS}}_{\mathsf{UV}}$ *such that*

$$\mathsf{Adv}^{\mathsf{UV}}_{\mathsf{SS}}(\mathcal{A}^{\mathsf{SS}}_{\mathsf{UV}}) \leq \mathsf{Adv}^{\mathsf{UV}}_{\mathsf{DS}}(\mathcal{A}^{\mathsf{DS}}_{\mathsf{UV}}).$$

The proof of Proposition 2 for $\mathsf{SS} = \mathsf{StE}$ is fairly straightforward and is given in Appendix D.1.

Proposition 3. *Let* $\mathsf{DS} = \mathsf{Ed25519\text{-}DS}[k, p, \mathbb{G}, \mathbb{G}_p, \mathbf{B}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3]$ *be the digital signature scheme built from some* $k, p, \mathbb{G}, \mathbb{G}_p, \mathbf{B}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$ *as specified in Construction 6. Let* $\mathcal{A}_{\mathsf{UV}}$ *be any adversary against the* UV *security of* DS*. Let* H_3 *be modeled as a random oracle and* q_{H} *be the number of random oracle queries that* $\mathcal{A}_{\mathsf{UV}}$ *makes. Then*

$$\mathsf{Adv}^{\mathsf{UV}}_{\mathsf{Ed25519}}(\mathcal{A}_{\mathsf{UV}}) \leq (2 \cdot q_{\mathrm{H}})/p + q_{\mathrm{H}}^2/p.$$

We now show that OAE security of a symmetric signcryption scheme $\mathsf{SS} \in \{\mathsf{StE}, \mathsf{EtS}, \mathsf{SP}, \mathsf{BM}\}$ with respect to $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$ follows from its OAE security with respect to $\mathsf{func}^{\perp}_{\mathsf{out}}$ and its UV security.

Scheme SS	Security notion OAE^*	Triviality predicate $\mathsf{pred}_{\mathsf{trivial}}$
$\mathsf{StE} = \mathsf{SIGN\text{-}THEN\text{-}ENCRYPT\text{-}SS}[\mathsf{DS}, \mathsf{NE}]$	OAE	$\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$
$\mathsf{EtS} = \mathsf{ENCRYPT\text{-}THEN\text{-}SIGN\text{-}SS}[\mathsf{NE}, \mathsf{DS}]$	OAE	$\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$
$\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}]$	bwOAE	$\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$
$\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{H}, \mathsf{DS}, \mathsf{NE}]$	$\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$	$\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$

Table 1: Table specifying the OAE^* security notion and the predicate $\mathsf{pred}_{\mathsf{trivial}}$ for different values of SS. Here, $\mathsf{StE} = \mathsf{SIGN\text{-}THEN\text{-}ENCRYPT\text{-}SS}[\mathsf{DS}, \mathsf{NE}]$ and $\mathsf{EtS} = \mathsf{ENCRYPT\text{-}THEN\text{-}SIGN\text{-}SS}[\mathsf{NE}, \mathsf{DS}]$ are the symmetric signcryption schemes built from some DS, NE as specified in Construction 1 and Construction 2 respectively. The scheme $\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{NE}, \mathsf{H}, \mathsf{SP}]$ is the symmetric signcryption scheme built from $\mathcal{M} = \{0,1\}^*$ and some $\mathsf{NE}, \mathsf{H}, \mathsf{SP}$ as specified in Construction 3. Finally, the scheme $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{H}, \mathsf{DS}, \mathsf{NE}]$ is the symmetric signcryption scheme built from some $\mathsf{H}, \mathsf{DS}, \mathsf{NE}$ as specified in Construction 4.

Proposition 4. *Let* $\mathsf{StE}, \mathsf{EtS}, \mathsf{BM},$ *and* SP *be the symmetric signcryption schemes defined in Constructions 1, 2, 3 and 4 respectively. 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. 8. Let* $\mathsf{func}^{\perp}_{\mathsf{out}}$ *and* $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}} := \mathsf{func}^{\mathsf{silence\text{-}with\text{-}m_1}}_{\mathsf{out}}[\mathsf{pred}_{\mathsf{trivial}}]$ *be the output-guarding functions as defined in Fig. 9. Let* $\mathsf{SS} \in \{\mathsf{StE}, \mathsf{EtS}, \mathsf{SP}, \mathsf{BM}\}$*, and* OAE^* *and* $\mathsf{pred}_{\mathsf{trivial}}$ *be as defined in Table 1. Let* $\mathcal{A}_{\mathsf{OAE}^*}$ *be any adversary against the* OAE^* *security of* SS *with respect to* $\mathsf{pred}_{\mathsf{trivial}}$ *and* $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$*. Then we can build adversaries* $\mathcal{A}_{\mathsf{UV}}$ *and* $\mathcal{A}^{\perp}_{\mathsf{OAE}^*}$ *such that*

$$\mathsf{Adv}^{\mathsf{OAE}^*}_{\mathsf{SS},\mathsf{pred}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{OAE}^*}) \leq \mathsf{Adv}^{\mathsf{UV}}_{\mathsf{SS}}(\mathcal{A}_{\mathsf{UV}}) + \mathsf{Adv}^{\mathsf{OAE}^*}_{\mathsf{SS},\mathsf{pred}_{\mathsf{trivial}},\mathsf{func}^{\perp}_{\mathsf{out}}}(\mathcal{A}^{\perp}_{\mathsf{OAE}^*}).$$

The proof of Proposition 4 is in Appendix D.3.

6.5 Instantiation and Interpretation

In this section, we summarize the analysis of $\mathsf{BoxMessage}$ that we provided in Sections 6.1 to 6.3. For each of the two core security notions in our paper, we state a separate corollary about what is achieved by $\mathsf{BoxMessage}$. We choose to present our corollaries in a precise and formal way, meaning they do not simplify or approximate any of the advantage terms from our analysis in the prior sections. However, instead of using the abstract schemes, we phrase our corollaries with respect to the real-world primitives that are used to instantiate $\mathsf{BoxMessage}$ in Keybase, as described in Section 5.

In the following, we show that our analysis of the in-group unforgeability of $\mathsf{BoxMessage}$ in Section 6.1 reduces the IUF security of $\mathsf{BoxMessage}$ with respect to the predicate $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ to the collision resistance of the $\mathsf{SHA\text{-}256}$ and $\mathsf{SHA\text{-}512}$ hash functions and the SUFCMA security of the $\mathsf{Ed25519}$ signature scheme. Recall that in the IUF security game, we use ciphertext triviality predicates to detect ciphertexts that were trivially forwarded from the SigEnc oracle to the VerDec oracle. In particular, the predicate $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ captures the strong unforgeability of ciphertexts by checking whether $z \in C$. We note here that the following result holds for any choice of the message space $\mathcal{M}$. Moreover, the SUFCMA security of Keybase's implementation of $\mathsf{Ed25519}$ is shown in [BCJZ21].

Corollary 1. *Let* $\mathsf{SHA\text{-}256}$, $\mathsf{SHA\text{-}512}$, $\mathsf{Ed25519}$, $\mathsf{XSalsa20\text{-}Poly1305}$ *be the standard real-world schemes used in Keybase. Let* $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{SHA\text{-}512}, \mathsf{Ed25519}, \mathsf{XSalsa20\text{-}Poly1305}]$ *be the symmetric signcryption scheme as defined in Construction 4. Let* $\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{XSalsa20\text{-}Poly1305}, \mathsf{SHA\text{-}256}, \mathsf{SP}]$ *be the symmetric signcryption scheme as defined in Construction 3, instantiated for any message space* $\mathcal{M}$*. Let* $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 8. 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* $\mathcal{A}_{\mathsf{SUFCMA}}$*,* $\mathcal{A}_{\mathsf{CR\text{-}of\text{-}SHA512}}$*, and* $\mathcal{A}_{\mathsf{CR\text{-}of\text{-}SHA256}}$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{BM},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}BM}}) \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{Ed25519}}(\mathcal{A}_{\mathsf{SUFCMA}}) &+ \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{SHA\text{-}512}}(\mathcal{A}_{\mathsf{CR\text{-}of\text{-}SHA512}}) \\ &+ \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{SHA\text{-}256}}(\mathcal{A}_{\mathsf{CR\text{-}of\text{-}SHA256}}).\end{aligned}$$

Next, we show that our analysis of the out-group AE security of $\mathsf{BoxMessage}$ in Sections 6.2 and 6.3 reduces the bwOAE security of $\mathsf{BoxMessage}$ with respect to the predicate $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ and the output-guarding function $\mathsf{func}^{\perp}_{\mathsf{out}}$ to the key-recovery and AEAD security of $\mathsf{XSalsa20\text{-}Poly1305}$, the sparsity of $\mathsf{Ed25519}$ with respect to the message space $\mathcal{M}$ underlying $\mathsf{BoxMessage}$, and the collision resistance of $\mathsf{SHA\text{-}256}$. Recall that bwOAE is a variant of the OAE security notion that modifies the group-nonce uniqueness condition of the latter. In the bwOAE game, we use ciphertext triviality predicates to detect ciphertexts that were trivially forwarded from the SigEnc oracle to the VerDec oracle and the function $\mathsf{func}^{\perp}_{\mathsf{out}}$ to appropriately respond with $\perp$ when the VerDec oracle is queried on such forwarded ciphertexts.

Corollary 2. *Let* $\mathsf{SHA\text{-}256}$, $\mathsf{SHA\text{-}512}$, $\mathsf{Ed25519}$, $\mathsf{XSalsa20\text{-}Poly1305}$ *be the standard real-world schemes used in Keybase. Let* $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{SHA\text{-}512}, \mathsf{Ed25519}, \mathsf{XSalsa20\text{-}Poly1305}]$ *be the symmetric signcryption scheme as defined in Construction 4. Let* $\mathsf{BM} = \mathsf{BOX\text{-}MESSAGE\text{-}SS}[\mathcal{M}, \mathsf{XSalsa20\text{-}Poly1305}, \mathsf{SHA\text{-}256}, \mathsf{SP}]$ *be the symmetric signcryption scheme as defined in Construction 3, instantiated for the message space* $\mathcal{M}$ *that contains application-layer messages encoded in the* $\mathsf{MessagePack}$ *format. Let* $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 8. Let* $\mathsf{func}^{\perp}_{\mathsf{out}}$ *be the output-guarding function as defined in Fig. 9. 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* $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$*, making at most* $q_{\mathrm{NewHonGroup}}$ *queries to its* NewHonGroup *oracle. Then we can build adversaries* $\mathcal{A}_{\mathsf{KR}}$*,* $\mathcal{A}_{\mathsf{AEAD}}$*,* $\mathcal{A}_{\mathsf{SPARSE}}$*, and* $\mathcal{A}_{\mathsf{CR\text{-}of\text{-}SHA256}}$ *such that*

$$\begin{aligned}&\mathsf{Adv}^{\mathsf{bwOAE}}_{\mathsf{BM},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}},\mathsf{func}^{\perp}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{bwOAE\text{-}of\text{-}BM}}) \\ &\quad\leq 2 \cdot \mathsf{Adv}^{\mathsf{KR}}_{\mathsf{XSalsa20\text{-}Poly1305}}(\mathcal{A}_{\mathsf{KR}}) + \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{XSalsa20\text{-}Poly1305}}(\mathcal{A}_{\mathsf{AEAD}}) + \frac{q^2_{\mathrm{NewHonGroup}}}{2^{257}} \\ &\quad+ \mathsf{Adv}^{\mathsf{SPARSE}}_{\mathsf{Ed25519},\mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}}) + \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{SHA\text{-}256}}(\mathcal{A}_{\mathsf{CR\text{-}of\text{-}SHA256}}).\end{aligned}$$

We emphasize that for this result, we model the (modularly reduced) instances of $\mathsf{SHA\text{-}512}$ within $\mathsf{Ed25519}$ as a random oracle. We also note from our analysis of the sparsity of $\mathsf{Ed25519}$ with respect to the message space $\mathcal{M}$ in Appendix C.1 that we expect the $\mathsf{Adv}^{\mathsf{SPARSE}}_{\mathsf{Ed25519},\mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}})$ term in the bound above to be very small for the parameters that $\mathsf{BoxMessage}$ is instantiated within Keybase.

7 Conclusions

Combining Theorem 2 with Theorem 3 and Theorem 4 with Theorem 5 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 in 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

ACDJ22. Martin R Albrecht, Sofía Celi, Benjamin Dowling, and Daniel Jones. Practically-exploitable cryptographic vulnerabilities in matrix. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1419–1436. IEEE Computer Society, 2022.

ACDT21. Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Modular design of secure group messaging protocols and the security of MLS. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1463–1483. ACM Press, November 2021. doi:10.1145/3460120.3484820.

ADJ23. Martin Albrecht, Benjamin Dowling, and Daniel Jones. Device-oriented group messaging: A formal cryptographic analysis of matrix'core. In *IEEE S&P 2024*. 2023.

ADR02. Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 83–107. Springer, Heidelberg, April / May 2002. doi:10.1007/3-540-46035-7_6.

AJKL23. Joël Alwen, Jonas Janneck, Eike Kiltz, and Benjamin Lipp. The pre-shared key modes of hpke. In *Advances in Cryptology - ASIACRYPT 2023*, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-981-99-8736-8_11.

BBLW22. Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A. Wood. Hybrid Public Key Encryption. RFC 9180, February 2022. doi:10.17487/RFC9180.

BBR$^+$23. Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. RFC 9420, July 2023. doi:10.17487/RFC9420.

BCG23. David Balbás, Daniel Collins, and Phillip Gajland. Whatsupp with sender keys? analysis, improvements and security proofs. In *Advances in Cryptology - ASIACRYPT 2023*, pages 307–341, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-981-99-8733-7_10.

BCJZ21. Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. The provable security of Ed25519: Theory and practice. In *2021 IEEE Symposium on Security and Privacy*, pages 1659–1676. IEEE Computer Society Press, May 2021. doi:10.1109/SP40001.2021.00042.

BDD23. Mihir Bellare, Hannah Davis, and Zijing Di. Hardening signature schemes via derive-then-derandomize: Stronger security proofs for EdDSA. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 223–250. Springer, Heidelberg, May 2023. doi:10.1007/978-3-031-31368-4_9.

BDJR97. Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, October 1997. doi:10.1109/SFCS.1997.646128.

BDL$^+$11. Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 124–142. Springer, Heidelberg, September / October 2011. doi:10.1007/978-3-642-23951-9_9.

BDL$^+$12. Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, September 2012. doi:10.1007/s13389-012-0027-1.

Ber05. Daniel J. Bernstein. The poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *FSE 2005*, volume 3557 of *LNCS*, pages 32–49. Springer, Heidelberg, February 2005. doi:10.1007/11502760_3.

Ber08. Daniel J Bernstein. The salsa20 family of stream ciphers. In *New stream cipher designs: the eSTREAM finalists*, pages 84–97. Springer, 2008.

BK11. Mihir Bellare and Sriram Keelveedhi. Authenticated and misuse-resistant encryption of key-dependent data. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 610–629. Springer, Heidelberg, August 2011. doi:10.1007/978-3-642-22792-9_35.

BMT14. Mihir Bellare, Sarah Meiklejohn, and Susan Thomson. Key-versatile signatures and applications: RKA, KDM and joint enc/sig. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 496–513. Springer, Heidelberg, May 2014. doi:10.1007/978-3-642-55220-5_28.

BP02. Mihir Bellare and Adriana Palacio. GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 162–177. Springer, Heidelberg, August 2002. doi:10.1007/3-540-45708-9_11.

BPS07. Michael Backes, Birgit Pfitzmann, and Andre Scedrov. Key-dependent message security under active attacks - BRSIM/UC-soundness of symbolic encryption with key cycles. In Andrei Sabelfeld, editor, *CSF 2007 Computer Security Foundations Symposium*, pages 112–124. IEEE Computer Society Press, 2007. doi:10.1109/CSF.2007.23.

BR06. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006. doi:10.1007/11761679_25.

BRS03. John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 62–75. Springer, Heidelberg, August 2003. doi:10.1007/3-540-36492-7_6.

BS20. Mihir Bellare and Igors Stepanovs. Security under message-derived keys: Signcryption in iMessage. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 507–537. Springer, Heidelberg, May 2020. doi:10.1007/978-3-030-45727-3_17.

CMRR23. Lily Chen, Dustin Moody, Andrew Regenscheid, and Angela Robinson. Digital signature standard (dss), 2023-02-02 05:02:00 2023. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=935202, doi:https://doi.org/10.6028/NIST.FIPS.186-5.

Fur. Sadayuki Furuhashi. Messagepack. https://msgpack.org/.

Keya. Keybase. Keybase Book. https://book.keybase.io/.

Keyb. Keybase. Keybase Book – Chat – Crypto. https://github.com/keybase/book-content/blob/master/D-docs/04-chat/01-crypto.md?plain=1#L89-L93.

Keyc. Keybase. Keybase client. https://github.com/keybase/client.

Keyd. Keybase. Keybase client – boxer.go – BoxMessage. https://github.com/keybase/client/blob/v6.2.2/go/chat/boxer.go/#L1564-L1566.

Keye. Keybase. Keybase client – codec.go – Design Notes. https://github.com/keybase/client/blob/v6.2.2/go/chat/signencrypt/codec.go/#L95-L110.

Keyf. Keybase. Keybase client – codec.go – sealPacket. https://github.com/keybase/client/blob/v6.2.2/go/chat/signencrypt/codec.go/#L189-L190.

Keyg. Keybase. Keybase client – codec.go – sealPacket. https://github.com/keybase/client/blob/v6.2.2/go/chat/signencrypt/codec.go#L548-L549.

Keyh. Keybase. Keybase client – codec.go – sealPacket. https://github.com/keybase/client/blob/v6.2.2/go/chat/attachments/sender.go#L130-L133.

Keyi. Keybase. Keybase stats. https://web.archive.org/web/20200207065125/https://keybase.io/. Accessed 28 February 2024.

Mar14. Moxie Marlinspike. Private group messaging. https://signal.org/blog/private-groups/, May 2014.

NRS14. Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 257–274. Springer, Heidelberg, May 2014. doi:10.1007/978-3-642-55220-5_15.

Rog04. Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 348–359. Springer, Heidelberg, February 2004. doi:10.1007/978-3-540-25937-4_22.

Rog06. Phillip Rogaway. Formalizing human ignorance. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06*, volume 4341 of *LNCS*, pages 211–228. Springer, Heidelberg, September 2006.

RPF19. Keegan Ryan, Thomas Pornin, and Shawn Fitzgerald. Keybase protocol security review. https://keybase.io/docs-assets/blog/NCC_Group_Keybase_KB2018_Public_Report_2019-02-27_v1.3.pdf, Feb 2019.

RS06. Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, Heidelberg, May / June 2006. doi:10.1007/11761679_23.

Shr04. Tom Shrimpton. 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.

Wha23. WhatsApp. Whatsapp encryption overview: Technical white paper. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf, Sep 2023.

WPBB23. Théophile Wallez, Jonathan Protzenko, Benjamin Beurdouche, and Karthikeyan Bhargavan. TreeSync: Authenticated group management for messaging layer security. In *32nd USENIX Security Symposium*, pages 1217–1233, Anaheim, CA, August 2023. USENIX Association.

Zhe97. Yuliang Zheng. Digital signcryption or how to achieve cost(signature & encryption) $\ll$ cost(signature) + cost(encryption). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 165–179. Springer, Heidelberg, August 1997. doi:10.1007/BFb0052234.

Zoo20. Zoom. Zoom acquires keybase and announces goal of developing the most broadly used enterprise end-to-end encryption offering. https://blog.zoom.us/zoom-acquires-keybase-and-announces-goal-of-developing-the-most-broadly-used-enterprise-end-to-end-encryption-offering/, May 2020.

A Proofs for Sign-Then-Encrypt Based Symmetric Signcryption

A.1 In-Group Unforgeability of StE (Proof of Theorem 1)

```
Games $\mathcal{G}_0$–$\mathcal{G}_2$
$\mathcal{A}_{\mathsf{IUF}}^{\text{U,G,SigEnc,VerDec}}$ ; Return win

SigEnc$(g, u, n, m, ad)$
require $\mathsf{K}[g] \neq \bot$ and $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$m_s \leftarrow \langle g, n, m, ad \rangle$ ; $s \leftarrow\!\!\$\ \mathsf{DS.Sig}(\mathsf{sk}[u], m_s)$
$S \leftarrow S \cup \{(u, m_s, s)\}$ ; $m_e \leftarrow s \,\|\, m$ ; $ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$
$c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_e, ad_{\mathsf{NE}})$
$C \leftarrow C \cup \{((g, u, n, m, ad), c)\}$ ; Return $c$

NewHonUser$(u)$
NewHonGroup$(g, \mathsf{users})$
ExposeUser$(u)$
ExposeGroup$(g)$
NewCorrUser$(u, sk, vk)$
NewCorrGroup$(g, K, \mathsf{users})$
// These oracles are identical to the corresponding
// oracles in the IUF game for StE as per Fig. 7.
```

```
VerDec$(g, u, n, c, ad)$
require $\mathsf{K}[g] \neq \bot$ and $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$ ; $m_e \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c, ad_{\mathsf{NE}})$
If $m_e = \bot$ then return $\bot$
$s \,\|\, m \leftarrow m_e$   // s.t. $|s| = \mathsf{DS.sl}$, $|m| \geq 0$
$m_s \leftarrow \langle g, n, m, ad \rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], m_s, s)$ then return $\bot$
$z \leftarrow ((g, u, n, m, ad), c)$
If $z \in C$ then return $m$
If $\neg\mathsf{user\_is\_corrupt}[u]$ then
  If $(u, m_s, s) \notin S$ then
    $\mathsf{bad}_0 \leftarrow \mathsf{true}$   // $\mathcal{A}_{\mathsf{SUFCMA}}$ breaks SUFCMA of DS.
    $\mathsf{win} \leftarrow \mathsf{true}$   // $\mathcal{G}_{[0,1)}$
  Else   // $(u, m_s, s) \in S$
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$   // Contradicts the tidiness of NE.
    $\mathsf{win} \leftarrow \mathsf{true}$   // $\mathcal{G}_{[0,2)}$
Return $m$
```

```
Adversary $\mathcal{A}_{\mathsf{SUFCMA}}^{\text{U,Sign}}$
$\mathcal{A}_{\mathsf{IUF}}^{\text{SimU,SimG,SimSigEnc,SimVerDec}}$

SimSigEnc$(g, u, n, m, ad)$
require $\mathsf{K}[g] \neq \bot$ and $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$m_s \leftarrow \langle g, n, m, ad \rangle$ ; $s \leftarrow \text{Sign}(u, m_s)$
$S \leftarrow S \cup \{(u, m_s, s)\}$ ; $m_e \leftarrow s \,\|\, m$ ; $ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$
$c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_e, ad_{\mathsf{NE}})$
$C \leftarrow C \cup \{((g, u, n, m, ad), c)\}$ ; Return $c$

SimNewHonGroup$(g, \mathsf{users})$
SimExposeGroup$(g)$
SimNewCorrGroup$(g, K, \mathsf{users})$
// These oracles are identical to the corresponding
// oracles in the IUF game for StE as per Fig. 7.
```

```
SimVerDec$(g, u, n, c, ad)$
require $\mathsf{K}[g] \neq \bot$ and $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$ ; $m_e \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c, ad_{\mathsf{NE}})$
If $m_e = \bot$ then return $\bot$
$s \,\|\, m \leftarrow m_e$   // s.t. $|s| = \mathsf{DS.sl}$, $|m| \geq 0$
$m_s \leftarrow \langle g, n, m, ad \rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], m_s, s)$ then return $\bot$
$z \leftarrow ((g, u, n, m, ad), c)$
If $z \in C$ then return $m$
If $\neg\mathsf{user\_is\_corrupt}[u]$ then
  If $(u, m_s, s) \notin S$ then
    $z \leftarrow (u, m_s, s)$
    abort$(z)$
Return $m$
```

```
SimNewHonUser$(u)$
require $\mathsf{sk}[u] = \mathsf{vk}[u] = \bot$
$\mathsf{sk}[u] \leftarrow$ "dummy"
$\mathsf{vk}[u] \leftarrow$ NewHonUser$(u)$
Return $\mathsf{vk}[u]$
```

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

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

Fig. 20: **Top pane:** Games $\mathcal{G}_0$–$\mathcal{G}_2$ for the proof of Theorem 1. The code highlighted in gray was added by expanding the symmetric signcryption algorithms StE.SigEnc, StE.VerDec and the ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ in game $\mathcal{G}^{\mathsf{IUF}}_{\mathsf{StE},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}})$. The code added for the transitions between games is highlighted in green. **Bottom pane:** Adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ for the proof of Theorem 1. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_0$.

Overview. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_2$ in Fig. 20. We establish the following claims.

- $\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{StE},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) = \Pr[\mathcal{G}_0]$
- $\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1] \le \Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$
- $\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \le \Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$
- $\Pr[\mathcal{G}_2] = 0$
- $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \le \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})$
- $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}] = 0$

By combining the above, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{StE},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) = \Pr[\mathcal{G}_0] &= \sum_{i=0}^{1}(\Pr[\mathcal{G}_i] - \Pr[\mathcal{G}_{i+1}]) + \Pr[\mathcal{G}_2] \\ &\le \Pr[\mathsf{bad}_0^{\mathcal{G}_0}] + \Pr[\mathsf{bad}_1^{\mathcal{G}_1}] \\ &\le \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}}).\end{aligned}$$

We justify our claims in the following paragraphs.

Game $\mathcal{G}_0$. Game $\mathcal{G}_0$ is functionally equivalent to game $\mathcal{G}^{\mathsf{IUF}}_{\mathsf{StE},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}})$. The former expands multiple instructions in the latter; the expanded code is marked in gray. Game $\mathcal{G}_0$ also contains newly added code that is highlighted in green and does not affect its functionality; this code will be used for transitions between games. In particular, the `if-then-else` statement that was added in oracle VerDec sets the win flag in each of its conditional branches. It follows that

$$\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{StE},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) = \Pr[\mathcal{G}_0].$$

Transitions from $\mathcal{G}_0$ to $\mathcal{G}_2$. Let $i \in \{0, 1\}$. Games $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ are identical until bad_i is set, therefore

$$\Pr[\mathcal{G}_i] - \Pr[\mathcal{G}_{i+1}] \le \Pr[\mathsf{bad}_i^{\mathcal{G}_i}].$$

Each of these transitions removes an instruction that sets the win flag. The win flag can no longer be set in game $\mathcal{G}_2$, making it impossible for $\mathcal{A}_{\mathsf{IUF}}$ to win. We have

$$\Pr[\mathcal{G}_2] = 0.$$

Bounding $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$. In Fig. 20 we build an adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ against the SUFCMA security of DS. It simulates game $\mathcal{G}_0$ for adversary $\mathcal{A}_{\mathsf{IUF}}$ as follows. When adversary $\mathcal{A}_{\mathsf{IUF}}$ calls the user oracles in the IUF security game, adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ responds by using its own user oracles from the SUFCMA security game. In response to $\mathcal{A}_{\mathsf{IUF}}$'s calls to the group oracles in the IUF security game, we have $\mathcal{A}_{\mathsf{SUFCMA}}$ respond by honestly sampling and managing its own NE keys. When $\mathcal{A}_{\mathsf{IUF}}$ calls the SigEnc oracle, adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ can now simulate it by using its Sign oracle. Finally, $\mathcal{A}_{\mathsf{IUF}}$'s calls to the VerDec oracle can be trivially simulated by $\mathcal{A}_{\mathsf{SUFCMA}}$ because this oracle does not use DS signing keys. Observe that bad_0 can be set in game $\mathcal{G}_0$ only after VerDec confirms the following three statements to be true: $\mathsf{DS.Ver}(\mathsf{vk}[u], m_s, s)$, $\neg\mathsf{user\_is\_corrupt}[u]$, and $(u, m_s, s) \notin S$. These are exactly the three winning conditions used in the SUFCMA security game, so $\mathcal{A}_{\mathsf{SUFCMA}}$ halts with $\textbf{abort}(u, m_s, s)$ whenever it detects bad_0 being set in the simulated IUF security game. We have

$$\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \le \Pr[\mathcal{G}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})] = \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}}).$$

Bounding $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$. Now consider what conditions need to be met for bad_1 to be set in game $\mathcal{G}_1$. This only happens when $z \notin C$ and $(u, m_s, s) \in S$ are simultaneously true. We can use the definitions of the encoding m_s and the tuple z in order to expand $(u, m_s, s) \in S$ into $((u, \langle g, n, m, ad\rangle), s) \in S$, and $z \notin C$ into $((g, u, n, m, ad), c) \notin C$. The former means that one of the prior calls to the SigEnc oracle produced the ciphertext $c' = \mathsf{NE.Enc}(\mathsf{K}[g], n, m_e, ad_{\mathsf{NE}})$ for $m_e = s \,\|\, m$ and $ad_{\mathsf{NE}} = \langle u, ad\rangle$. But the latter implies that in the current call to the VerDec oracle a *distinct* ciphertext $c \neq c'$ was used to recover $m_e = \mathsf{NE.Dec}(\mathsf{K}[g], n, c, ad_{\mathsf{NE}})$. Together this means that $\mathsf{NE.Enc}(\mathsf{K}[g], n, \mathsf{NE.Dec}(\mathsf{K}[g], n, c, ad_{\mathsf{NE}}), ad_{\mathsf{NE}}) \neq c$, violating the assumed tidiness property of NE. We get a contradiction, and hence

$$\Pr[\mathsf{bad}_1^{\mathcal{G}_1}] = 0.$$

A.2 Out-group Authenticated Encryption of StE (Proof of Theorem 1)

Overview. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_2$ in Fig. 21. We establish the following claims.

```
Games $\mathcal{G}_0$–$\mathcal{G}_2$
$b \leftarrow\$ \{0,1\}$ ; $b' \leftarrow\$ \mathcal{A}_{\mathsf{OAE}}^{\textsc{U},\textsc{G},\textsc{SigEnc},\textsc{VerDec}}$ ; 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 $\bot$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$m_s \leftarrow \langle g, n, m_b, ad \rangle$ ; $s \leftarrow\$ \mathsf{DS.Sig}(\mathsf{sk}[u], m_s)$
$m_e \leftarrow s \,\|\, m_b$ ; $ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$
$c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_e, ad_{\mathsf{NE}})$
$N_0 \leftarrow N_0 \cup \{(g, u, n, m_0, ad)\}$
$N_1 \leftarrow N_1 \cup \{(g, u, n, m_1, ad)\}$
$C \leftarrow C \cup \{((g, u, n, m_b, ad), c)\}$
$Q \leftarrow Q \cup \{((g, u, n, m_0, m_1, ad), c)\}$
$W[g, u, n, c, ad] \leftarrow m_1$ ; 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 $u \in \mathsf{members}[g]$
If $W[g, u, n, c, ad] \neq \bot$ then return $\bot$          // $\mathcal{G}_{[1,\infty)}$
$ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$ ; $m_e \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c, ad_{\mathsf{NE}})$
If $m_e = \bot$ then return $\bot$
$s \,\|\, m \leftarrow m_e$   // s.t. $|s| = \mathsf{DS.sl}$, $|m| \geq 0$
$m_s \leftarrow \langle g, n, m, ad \rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], m_s, s)$ then return $\bot$
$z \leftarrow ((g, u, n, m, ad), c)$                         // $\mathcal{G}_{[0,2)}$
If $z \in C$ then return $\bot$                              // $\mathcal{G}_{[0,2)}$
If $b = 0$ then return $\bot$ else return $m$

$\textsc{NewHonUser}(u)$
$\textsc{NewHonGroup}(g, \mathsf{users})$
$\textsc{ExposeUser}(u)$
$\textsc{ExposeGroup}(g)$
$\textsc{NewCorrUser}(u, sk, vk)$
$\textsc{NewCorrGroup}(g, K, \mathsf{users})$
// These oracles are identical to the corresponding
// oracles in the OAE game for StE as per Fig. 7.
```

```
Adversary $\mathcal{A}_{\mathsf{AEAD}}^{\textsc{G},\textsc{Enc},\textsc{Dec}}$
$b' \leftarrow\$ \mathcal{A}_{\mathsf{OAE}}^{\textsc{SimU},\textsc{SimG},\textsc{SimSigEnc},\textsc{SimVerDec}}$ ; Return $b'$

$\textsc{SimSigEnc}(g, u, n, m_0, m_1, ad)$
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 $\bot$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
If $m_0 = m_1$ then
    $m_s \leftarrow \langle g, n, m_0, ad \rangle$ ; $s \leftarrow\$ \mathsf{DS.Sig}(\mathsf{sk}[u], m_s)$
    $m_e \leftarrow s \,\|\, m_0$ ; $ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$
    $c \leftarrow \textsc{Enc}(g, n, m_e, m_e, ad_{\mathsf{NE}})$
Else   // $m_0 \neq m_1$
    $m_{s,0} \leftarrow \langle g, n, m_0, ad \rangle$ ; $s_0 \leftarrow\$ \mathsf{DS.Sig}(\mathsf{sk}[u], m_{s,0})$
    $m_{s,1} \leftarrow \langle g, n, m_1, ad \rangle$ ; $s_1 \leftarrow\$ \mathsf{DS.Sig}(\mathsf{sk}[u], m_{s,1})$
    $m_{e,0} \leftarrow s_0 \,\|\, m_0$ ; $m_{e,1} \leftarrow s_1 \,\|\, m_1$ ; $ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$
    $c \leftarrow \textsc{Enc}(g, n, m_{e,0}, m_{e,1}, ad_{\mathsf{NE}})$
$N_0 \leftarrow N_0 \cup \{(g, u, n, m_0, ad)\}$
$N_1 \leftarrow N_1 \cup \{(g, u, n, m_1, ad)\}$
$W[g, u, n, c, ad] \leftarrow m_1$ ; Return $c$
```

```
$\textsc{SimVerDec}(g, u, n, c, ad)$
require $\mathsf{K}[g] \neq \bot$ and $\neg\mathsf{group\_is\_corrupt}[g]$
require $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
If $W[g, u, n, c, ad] \neq \bot$ then return $\bot$
$ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$ ; $m_e \leftarrow \textsc{Dec}(g, n, c, ad_{\mathsf{NE}})$
If $m_e = \bot$ then return $\bot$
$\mathbf{abort}(1)$   // $m_e \neq \bot$ possible only when $b = 1$.

$\textsc{SimNewHonUser}(u)$
$\textsc{SimExposeUser}(u)$
$\textsc{SimNewCorrUser}(u, sk, vk)$
// These oracles are identical to the corresponding
// oracles in the OAE game for StE as per Fig. 7.
```

```
$\textsc{SimNewHonGroup}(g, \mathsf{users})$
require $\mathsf{K}[g] = \bot$
$\mathsf{K}[g] \leftarrow$ “dummy”
$\textsc{NewHonGroup}(g)$
$\mathsf{members}[g] \leftarrow \mathsf{users}$
```

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

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

Fig. 21: **Top pane:** Games $\mathcal{G}_0$–$\mathcal{G}_2$ for the proof of Theorem 1. The code highlighted in gray was added by expanding the symmetric signcryption algorithms $\mathsf{StE.SigEnc}$, $\mathsf{StE.VerDec}$ and the ciphertext-triviality predicate $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$ in game $\mathcal{G}_{\mathsf{StE},\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}},\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}}^{\mathsf{OAE}}(\mathcal{A}_{\mathsf{OAE}})$. The code added for the transitions between games is highlighted in green. **Bottom pane:** Adversary $\mathcal{A}_{\mathsf{AEAD}}$ for the proof of Theorem 1. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_2$.

- $\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{StE},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{OAE}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1$
- $\Pr[\mathcal{G}_0] = \Pr[\mathcal{G}_1]$
- $\Pr[\mathcal{G}_1] = \Pr[\mathcal{G}_2]$
- $2 \cdot \Pr[\mathcal{G}_2] - 1 \leq \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}})$

By combining the above, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{StE},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{OAE}}) &= 2 \cdot \Pr[\mathcal{G}_0] - 1 \\ &= 2 \cdot \Pr[\mathcal{G}_2] - 1 \leq \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}}).\end{aligned}$$

We justify our claims in the following paragraphs.

GAME $\mathcal{G}_0$. Game $\mathcal{G}_0$ is functionally equivalent to game $\mathcal{G}^{\mathsf{OAE}}_{\mathsf{StE},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{OAE}})$. The former expands multiple instructions in the latter; the expanded code is marked in gray. Game $\mathcal{G}_0$ also contains newly added code that is highlighted in green and does not affect its functionality; this code will be used for transitions between games. It follows that

$$\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{StE},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{OAE}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1.$$

TRANSITION FROM $\mathcal{G}_0$ TO $\mathcal{G}_1$. We now explain how game $\mathcal{G}_1$ differs from game $\mathcal{G}_0$, and will we argue that these two games are functionally equivalent. Game $\mathcal{G}_1$ maintains a record of all ciphertexts that were previously returned by its SIGENC oracle. When such a ciphertext gets subsequently passed as an input to the VERDEC oracle (with the matching values of g, u, n, ad), game $\mathcal{G}_1$ immediately returns $\bot$. To implement this functionality, game $\mathcal{G}_1$ uses a table W that maps (g, u, n, c, ad) to m_1. We claim that

$$\Pr[\mathcal{G}_0] = \Pr[\mathcal{G}_1].$$

In order to justify this transition, let us look at what happens in game $\mathcal{G}_0$ when an honestly produced ciphertext is forwarded from the SIGENC oracle to the VERDEC oracle. Consider an arbitrary SIGENC query that took some $(g^*, u^*, n^*, m_0^*, m_1^*, ad^*)$ as input to return a ciphertext c^*. At the end of this SIGENC call, the sets N, C, and Q are updated to insert new elements that are created from the SIGENC's input $(g^*, u^*, n^*, m_0^*, m_1^*, ad^*)$ and output c^*; our analysis below will crucially rely on these sets containing the newly added elements. Now consider what happens if the VERDEC oracle is queried on $(g^*, u^*, n^*, c^*, ad^*)$ at any point later on. If the condition checking $\neg\mathsf{group\_is\_corrupt}[g]$ passes, then the decryption correctness of NE and the correctness of DS jointly guarantee that VERDEC builds $z = ((g^*, u^*, n^*, m_b^*, ad^*), c^*)$. But exactly the same tuple was inserted into C during the earlier SIGENC call. It follows that $z \in C$ is true and the VERDEC oracle returns $\bot$. Therefore, the outputs of VERDEC are consistent across $\mathcal{G}_0$ and $\mathcal{G}_1$.

TRANSITION FROM $\mathcal{G}_1$ TO $\mathcal{G}_2$. Note that the condition checking $z \in C$ in oracle VERDEC of game $\mathcal{G}_1$ can never be reached. In particular, if the set C contains an element $z = ((g, u, n, m, ad), c)$ for any value of m, then $W[g, u, n, c, ad] \neq \bot$ must be true. The latter causes VERDEC to return $\bot$ right away. So in game $\mathcal{G}_2$ we remove the two lines of code that build z and check whether it belongs to C. Game $\mathcal{G}_2$ is functionally equivalent to game $\mathcal{G}_1$, so

$$\Pr[\mathcal{G}_1] = \Pr[\mathcal{G}_2].$$

BOUNDING $\Pr[\mathcal{G}_2]$. In Fig. 21 we build an adversary $\mathcal{A}_{\mathsf{AEAD}}$ against the AEAD security of NE that simulates game $\mathcal{G}_2$ for adversary $\mathcal{A}_{\mathsf{OAE}}$. Adversary $\mathcal{A}_{\mathsf{AEAD}}$ simulates the group oracles for $\mathcal{A}_{\mathsf{OAE}}$ by calling its own group oracles, and it simulates the user oracles for $\mathcal{A}_{\mathsf{OAE}}$ by sampling its own keys for DS.

Adversary $\mathcal{A}_{\mathsf{AEAD}}$ simulates the SIGENC oracle for $\mathcal{A}_{\mathsf{OAE}}$ by evaluating $\mathsf{NE.Enc}$ using its encryption oracle ENC. When SIGENC is queried with $m_0 = m_1$ as input, adversary $\mathcal{A}_{\mathsf{AEAD}}$ ensures that ENC is likewise called with both of its challenge-message arguments being equal. This maintains consistency of the challenge maps $\mathsf{chal}[\cdot]$ between the simulated game $\mathcal{G}_2$ and the AEAD security game of $\mathcal{A}_{\mathsf{AEAD}}$. When SIGENC is queried with $m_0 \neq m_1$ as input, note that $|m_{e,0}| = |m_{e,1}|$ is guaranteed to hold whenever $|m_0| = |m_1|$ does, because in Section 2.1 we require DS to produce signatures of a fixed length $\mathsf{DS.sl}$. Finally, the nonce-misuse condition in oracle SIGENC that checks $(g, n, m_*, ad) \notin N$ is equivalent to the corresponding condition in oracle ENC that checks $(g, u, n, m_*, ad) \notin N$, because $\mathcal{A}_{\mathsf{AEAD}}$ queries ENC on an ad field that uniquely encodes the u, ad fields of SIGENC.

Adversary $\mathcal{A}_{\mathsf{AEAD}}$ simulates the VERDEC oracle for $\mathcal{A}_{\mathsf{OAE}}$ by evaluating $\mathsf{NE.Dec}$ using its decryption oracle DEC. If DEC returns $m_e \neq \bot$ then the challenge bit in game $\mathcal{G}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}})$ is $b = 1$ and hence

$\mathcal{A}_{\mathsf{AEAD}}$ immediately calls **abort**(1) in order to halt and return 1 as output. Otherwise (i.e. $m_e = \bot$) adversary $\mathcal{A}_{\mathsf{AEAD}}$ returns $\bot$ as the output of the simulated VerDec oracle. Note that the AEAD security game defines Dec to return $\bot$ in response to the NE ciphertexts that were previously produced by its Enc oracle. This is equivalent to the case where a ciphertext is forwarded from SigEnc to VerDec; the use of the W map at the start of the simulated VerDec oracle ensures that Dec is never called on a replayed NE ciphertext.

Let d denote the challenge bit in game $\mathcal{G}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}})$, and let d' denote the corresponding output of adversary $\mathcal{A}_{\mathsf{AEAD}}$. If $d = 0$ then $\mathcal{A}_{\mathsf{AEAD}}$ perfectly simulates game $\mathcal{G}_2$ with challenge bit $b = 0$ for adversary $\mathcal{A}_{\mathsf{OAE}}$, and returns $d' = 1$ iff $\mathcal{A}_{\mathsf{OAE}}$ returns $b' = 1$. In particular, adversary $\mathcal{A}_{\mathsf{AEAD}}$ perfectly simulates the VerDec oracle because in game $\mathcal{G}_2$ this oracle always returns $\bot$ if $W[g, u, n, c, ad] = \bot$ and $b = 0$. We write $\Pr\left[d' = 1 \,\middle|\, d = 0\right] = \Pr\left[b' = 1 \,\middle|\, b = 0\right]$. If $d = 1$ then $\mathcal{A}_{\mathsf{AEAD}}$ perfectly simulates game $\mathcal{G}_2$ with challenge bit $b = 1$ for adversary $\mathcal{A}_{\mathsf{OAE}}$ *until* the first time $\mathsf{NE.Dec}$ produces non-$\bot$ in oracle VerDec of game $\mathcal{G}_2$. When the latter happens, $\mathcal{A}_{\mathsf{AEAD}}$ halts with $d' = 1$ regardless of whether $\mathcal{A}_{\mathsf{OAE}}$ would have returned $b' = 1$. We write $\Pr\left[d' = 1 \,\middle|\, d = 1\right] \geq \Pr\left[b' = 1 \,\middle|\, b = 1\right]$. It follows that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}}) &= \Pr\left[d' = 1 \,\middle|\, d = 1\right] - \Pr\left[d' = 1 \,\middle|\, d = 0\right] \\ &\geq \Pr\left[b' = 1 \,\middle|\, b = 1\right] - \Pr\left[b' = 1 \,\middle|\, b = 0\right] \\ &= 2 \cdot \Pr[\mathcal{G}_2] - 1.\end{aligned}$$

The latter equality relies on the standard fact that $\mathcal{A}_{\mathsf{OAE}}$'s advantage in game $\mathcal{G}_2$ can be expressed in two alternative ways that are equivalent.

B Proofs for the In-Group Unforgeability of Keybase

B.1 In-Group Unforgeability of BoxMessage (Proof of Theorem 2)

Overview. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_2$ in Fig. 22. We establish the following claims.

- $\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{BM},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}BM}}) = \Pr[\mathcal{G}_0]$
- $\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1] \leq \Pr[\mathsf{bad}^{\mathcal{G}_0}_0]$
- $\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \leq \Pr[\mathsf{bad}^{\mathcal{G}_1}_1]$
- $\Pr[\mathcal{G}_2] = 0$
- $\Pr[\mathsf{bad}^{\mathcal{G}_0}_0] \leq \mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{SP},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}})$
- $\Pr[\mathsf{bad}^{\mathcal{G}_1}_1] \leq \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{CR}})$

By combining the above, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{BM},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}BM}}) = \Pr[\mathcal{G}_0] &= \sum_{i=0}^{1}(\Pr[\mathcal{G}_i] - \Pr[\mathcal{G}_{i+1}]) + \Pr[\mathcal{G}_2] \\ &\leq \Pr[\mathsf{bad}^{\mathcal{G}_0}_0] + \Pr[\mathsf{bad}^{\mathcal{G}_1}_1] \\ &\leq \mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{SP},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}) + \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{CR}}).\end{aligned}$$

We justify our claims in the following paragraphs.

Game $\mathcal{G}_0$. Game $\mathcal{G}_0$ is functionally equivalent to game $\mathcal{G}^{\mathsf{IUF}}_{\mathsf{BM},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}$. The former expands multiple instructions in the latter; the expanded code is marked in gray. Game $\mathcal{G}_0$ also contains newly added code that is highlighted in green and does not affect its functionality; this code will be used for transitions between games. In particular, the `if-then-else` statement that was added in oracle VerDec sets the win flag in each of its conditional branches. It follows that

$$\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{BM},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}BM}}) = \Pr[\mathcal{G}_0].$$

Transitions from $\mathcal{G}_0$ to $\mathcal{G}_2$. Let $i \in \{0, 1\}$. Games $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ are identical until bad_i is set, therefore

$$\Pr[\mathcal{G}_i] - \Pr[\mathcal{G}_{i+1}] \leq \Pr[\mathsf{bad}^{\mathcal{G}_i}_i].$$

Each of these transitions removes an instruction that sets the win flag. In game $\mathcal{G}_2$ the win flag is only set when the conditions $\exists g' \colon ((g', u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}}) \in C_{\mathsf{SP}}$ and $n_{\mathsf{body}} \,\|\, c_{\mathsf{body}} = T[u, n_{\mathsf{header}}, c_{\mathsf{header}}, ad, g, h_{\mathsf{body}}]$ are satisfied. Since $m_{\mathsf{header}} = \langle ad, u, g, h_{\mathsf{body}} \rangle$, the former condition is necessarily true for

```
Games $\mathcal{G}_0$–$\mathcal{G}_2$
$\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}BM}}^{\mathrm{U,G,SigEnc,VerDec}}$ ; Return win

SigEnc$(g, u, n, m_{\mathsf{body}}, ad)$
require $\mathsf{K}[g] \neq \bot$ and $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$(n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n$
$c_{\mathsf{body}} \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n_{\mathsf{body}}, m_{\mathsf{body}})$
$h_{\mathsf{body}} \leftarrow \mathsf{H}(n_{\mathsf{body}} \| c_{\mathsf{body}})$ ; $m_{\mathsf{header}} \leftarrow \langle ad, u, g, h_{\mathsf{body}} \rangle$
$c_{\mathsf{header}} \leftarrow \mathsf{SP.SigEnc}(g, \mathsf{K}[g], u, \mathsf{sk}[u], n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon)$
$c \leftarrow (c_{\mathsf{body}}, c_{\mathsf{header}})$
$C_{\mathsf{SP}} \leftarrow C_{\mathsf{SP}} \cup \{((g, u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}})\}$
$T[u, n_{\mathsf{header}}, c_{\mathsf{header}}, ad, g, h_{\mathsf{body}}] \leftarrow n_{\mathsf{body}} \| c_{\mathsf{body}}$
$C \leftarrow C \cup \{((g, u, n, m_{\mathsf{body}}, ad), c)\}$ ; Return $c$

NewHonUser$(u)$
NewHonGroup$(g, \mathsf{users})$
ExposeUser$(u)$
ExposeGroup$(g)$
NewCorrUser$(u, sk, vk)$
NewCorrGroup$(g, K, \mathsf{users})$
// These oracles are identical to the corresponding
// oracles in the IUF game for BM as per Fig. 7.
```

```
VerDec$(g, u, n, c, ad)$
require $\mathsf{K}[g] \neq \bot$ and $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$(n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n$ ;  $(c_{\mathsf{body}}, c_{\mathsf{header}}) \leftarrow c$
$m_{\mathsf{header}} \leftarrow \mathsf{SP.VerDec}(g, \mathsf{K}[g], u, \mathsf{vk}[u], n_{\mathsf{header}}, c_{\mathsf{header}}, \varepsilon)$
$h_{\mathsf{body}} \leftarrow \mathsf{H}(n_{\mathsf{body}} \| c_{\mathsf{body}})$
If $m_{\mathsf{header}} \neq \langle ad, u, g, h_{\mathsf{body}} \rangle$ then return $\bot$
$m_{\mathsf{body}} \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n_{\mathsf{body}}, c_{\mathsf{body}})$
If $m_{\mathsf{body}} = \bot$ then return $\bot$
$z \leftarrow ((g, u, n, m_{\mathsf{body}}, ad), c)$ ; If $z \in C$ then return $m_{\mathsf{body}}$
If $\neg\mathsf{user\_is\_corrupt}[u]$ then
    If $\nexists g' : ((g', u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}}) \in C_{\mathsf{SP}}$ then
        $\mathsf{bad}_0 \leftarrow \mathsf{true}$   // $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$ breaks IUF of SP.
        win $\leftarrow$ true                                   // $\mathcal{G}_{[0,1)}$
    Else if $n_{\mathsf{body}} \| c_{\mathsf{body}} \neq T[u, n_{\mathsf{header}}, c_{\mathsf{header}}, ad, g, h_{\mathsf{body}}]$ then
        $\mathsf{bad}_1 \leftarrow \mathsf{true}$   // $\mathcal{A}_{\mathsf{CR}}$ breaks CR of H.
        win $\leftarrow$ true                                   // $\mathcal{G}_{[0,2)}$
    Else   // Unreachable in $\mathcal{G}_2$.
        win $\leftarrow$ true
Return $m_{\mathsf{body}}$
```

```
Adversary $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}^{\mathrm{U,G,SigEnc,VerDec}}$
$\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}BM}}^{\mathrm{SimU,SimG,SimSigEnc,SimVerDec}}$

SimSigEnc$(g, u, n, m_{\mathsf{body}}, ad)$
require $\mathsf{K}[g] \neq \bot$ and $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$(n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n$
$c_{\mathsf{body}} \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n_{\mathsf{body}}, m_{\mathsf{body}})$
$h_{\mathsf{body}} \leftarrow \mathsf{H}(n_{\mathsf{body}} \| c_{\mathsf{body}})$ ; $m_{\mathsf{header}} \leftarrow \langle ad, u, g, h_{\mathsf{body}} \rangle$
$c_{\mathsf{header}} \leftarrow$ SigEnc$(g, u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon)$
$c \leftarrow (c_{\mathsf{body}}, c_{\mathsf{header}})$
$C_{\mathsf{SP}} \leftarrow C_{\mathsf{SP}} \cup \{((g, u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}})\}$
$C \leftarrow C \cup \{((g, u, n, m_{\mathsf{body}}, ad), c)\}$ ; Return $c$
```

```
SimVerDec$(g, u, n, c, ad)$
require $\mathsf{K}[g] \neq \bot$ and $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$(n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n$ ; $(c_{\mathsf{body}}, c_{\mathsf{header}}) \leftarrow c$
$m_{\mathsf{header}} \leftarrow$ VerDec$(g, u, n_{\mathsf{header}}, c_{\mathsf{header}}, \varepsilon)$
$h_{\mathsf{body}} \leftarrow \mathsf{H}(n_{\mathsf{body}} \| c_{\mathsf{body}})$
If $m_{\mathsf{header}} \neq \langle ad, u, g, h_{\mathsf{body}} \rangle$ then return $\bot$
$m_{\mathsf{body}} \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n_{\mathsf{body}}, c_{\mathsf{body}})$
If $m_{\mathsf{body}} = \bot$ then return $\bot$
$z \leftarrow ((g, u, n, m_{\mathsf{body}}, ad), c)$ ; If $z \in C$ then return $m_{\mathsf{body}}$
If $\neg\mathsf{user\_is\_corrupt}[u]$ then
    If $\nexists g' : ((g', u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}}) \in C_{\mathsf{SP}}$ then
        $\mathsf{bad}_0 \leftarrow \mathsf{true}$   // $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$ breaks IUF of SP.
Return $m_{\mathsf{body}}$
```

```
SimNewHonUser$(u)$
require $\mathsf{sk}[u] = \mathsf{vk}[u] = \bot$
$\mathsf{sk}[u] \leftarrow$ "dummy"
$\mathsf{vk}[u] \leftarrow$ NewHonUser$(u)$
Return $\mathsf{vk}[u]$

SimNewHonGroup$(g, \mathsf{users})$
require $\mathsf{K}[g] = \bot$
NewHonGroup$(g, \mathsf{users})$
$\mathsf{K}[g] \leftarrow$ ExposeGroup$(g)$
$\mathsf{members}[g] \leftarrow \mathsf{users}$
```

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

SimExposeGroup$(g)$
require $\mathsf{K}[g] \neq \bot$
Return $\mathsf{K}[g]$
```

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

SimNewCorrGroup$(g, K, \mathsf{users})$
require $\mathsf{K}[g] = \bot$
NewCorrGroup$(g, K, \mathsf{users})$
$\mathsf{K}[g] \leftarrow K$ ; $\mathsf{members}[g] \leftarrow \mathsf{users}$
```

Fig. 22: **Top pane:** Games $\mathcal{G}_0$–$\mathcal{G}_2$ for the proof of Theorem 2. The code highlighted in gray was added by expanding the symmetric signcryption algorithms BM.SigEnc, BM.VerDec and the ciphertext-triviality predicate $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$ in game $\mathcal{G}_{\mathsf{BM},\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}}^{\mathsf{IUF}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}BM}})$. The code added for the transitions between games is highlighted in green. **Bottom pane:** Adversary $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$ for the proof of Theorem 2. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_1$.

$g' = g$, i.e. $((g, u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}}) \in C_{\mathsf{SP}}$. The two conditions combined with the correctness of NE guarantee that $((g, u, n, m_{\mathsf{body}}, ad), c) \in C$. This means that the ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ would have triggered for such a query prior to reaching the `if-then-else` statement. Hence, it is impossible to set the win flag in $\mathcal{G}_2$ and we have

$$\Pr[\mathcal{G}_2] = 0.$$

Bounding $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$. In Fig. 22 we build an adversary $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$ against the IUF security of SP with respect to the ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}$. It perfectly simulates game $\mathcal{G}_0$ for adversary $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}BM}}$. In particular, it simulates the responses to the user and group oracles using the corresponding oracles from its own IUF security game. Note that the IUF security game allows to expose all group keys, so every time an honest group key is created in the simulated game $\mathcal{G}_0$, adversary $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$ immediately exposes it. While simulating the SigEnc oracle for $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}BM}}$, it replaces calls to $\mathsf{SP.SigEnc}$ with its SigEnc oracle. Analogously, while simulating VerDec, it substitutes calls to $\mathsf{SP.VerDec}$ with its VerDec oracle. Finally, when the bad_0 flag is set in the simulated game $\mathcal{G}_0$, adversary $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$ wins by setting the win flag in the IUF security game it plays in. Note that the bad_0 flag is only set in $\mathcal{G}_0$ when $\neg\mathsf{user\_is\_corrupt}[u]$, and c_{header} successfully decrypts under $\mathsf{SP.VerDec}$ and $\nexists g' \colon ((g', u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}}) \in C_{\mathsf{SP}}$. These are the same as the winning conditions in the IUF security game of SP and so

$$\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq \mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{SP},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}).$$

Bounding $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$. In order to upper-bound $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$, consider an adversary $\mathcal{A}_{\mathsf{CR}}$ against the collision resistance of H that is defined as follows. Adversary $\mathcal{A}_{\mathsf{CR}}$ perfectly simulates game $\mathcal{G}_1$ for $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}BM}}$ until the bad_1 flag is set. To simulate the user oracles U and the group oracles G, adversary $\mathcal{A}_{\mathsf{CR}}$ itself samples and maintains all user keys and group keys. Whenever the bad_1 flag is set in VerDec, adversary $\mathcal{A}_{\mathsf{CR}}$ returns a pair of strings $(n_{\mathsf{body}} \,\|\, c_{\mathsf{body}}, T[u, n_{\mathsf{header}}, c_{\mathsf{header}}, ad, g, h_{\mathsf{body}}])$. Note that the table entry $T[u, n_{\mathsf{header}}, c_{\mathsf{header}}, ad, g, h_{\mathsf{body}}]$ is not empty because bad_1 could only be set once the following two conditions were satisfied in the current VerDec call: $\exists g' \colon ((g', u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}}) \in C_{\mathsf{SP}}$ and $m_{\mathsf{header}} = \langle ad, u, g, h_{\mathsf{body}} \rangle$. This means there was a prior SigEnc call where all of the values indexing the table T (as per the current VerDec call) were simultaneously in use. The two strings returned by $\mathcal{A}_{\mathsf{CR}}$ are distinct because a conditional statement in game $\mathcal{G}_1$ verifies this immediately before setting the bad_1 flag. We now show that these strings also produce the same hash value. First, observe that the table entry $T[u, n_{\mathsf{header}}, c_{\mathsf{header}}, ad, g, h_{\mathsf{body}}]$ was filled during a prior call to oracle SigEnc, with a value for which $h_{\mathsf{body}} = \mathsf{H}(T[u, n_{\mathsf{header}}, c_{\mathsf{header}}, ad, g, h_{\mathsf{body}}])$ is true. But the same hash was obtained in the current VerDec call upon evaluating $h_{\mathsf{body}} \leftarrow \mathsf{H}(n_{\mathsf{body}} \,\|\, c_{\mathsf{body}})$. So the strings returned by $\mathcal{A}_{\mathsf{CR}}$ produce a collision. It follows that

$$\Pr[\mathsf{bad}_1^{\mathcal{G}_1}] \leq \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{CR}}).$$

B.2 In-Group Unforgeability of SealPacket (Proof of Theorem 3)

Overview. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_3$ in Fig. 23. We establish the following claims.

- ○ $\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{SP},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}) = \Pr[\mathcal{G}_0]$
- ○ $\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$
- ○ $\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \leq \Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$
- ○ $\Pr[\mathcal{G}_2] - \Pr[\mathcal{G}_3] \leq \Pr[\mathsf{bad}_2^{\mathcal{G}_2}]$
- ○ $\Pr[\mathcal{G}_3] = 0$
- ○ $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})$
- ○ $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}] \leq \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{CR}})$
- ○ $\Pr[\mathsf{bad}_2^{\mathcal{G}_2}] = 0$

```
Games $\mathcal{G}_0$–$\mathcal{G}_3$
$\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}^{\textsc{U},\textsc{G},\textsc{SigEnc},\textsc{VerDec}}$ ; Return win

$\textsc{SigEnc}(g, u, n, m, ad)$   // $ad = \varepsilon$
require $\mathsf{K}[g] \neq \bot$ and $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$h \leftarrow \mathsf{H}(m)$ ; $m_s \leftarrow$ "Keybase-Chat-2" $\|$ $\langle \mathsf{K}[g], n, h \rangle$
$s \leftarrow \mathsf{DS.Sig}(\mathsf{sk}[u], m_s)$ ; $T[u, m_s, s] \leftarrow m$
$m_e \leftarrow s \,\|\, m$ ; $c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_e)$
$C \leftarrow C \cup \{((g, u, n, m, ad), c)\}$ ; Return $c$

$\textsc{NewHonUser}(u)$
$\textsc{NewHonGroup}(g, \mathsf{users})$
$\textsc{ExposeUser}(u)$
$\textsc{ExposeGroup}(g)$
$\textsc{NewCorrUser}(u, sk, vk)$
$\textsc{NewCorrGroup}(g, K, \mathsf{users})$
// These oracles are identical to the corresponding
// oracles in the IUF game for SP as per Fig. 7.
```

```
$\textsc{VerDec}(g, u, n, c, ad)$   // $ad = \varepsilon$
require $\mathsf{K}[g] \neq \bot$ and $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$m_e \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c)$
If $m_e = \bot$ then return $\bot$
$s \,\|\, m \leftarrow m_e$   // s.t. $|s| = \mathsf{DS.sl}$, $|m| \geq 0$
$h \leftarrow \mathsf{H}(m)$ ; $m_s \leftarrow$ "Keybase-Chat-2" $\|$ $\langle \mathsf{K}[g], n, h \rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], m_s, s)$ then return $\bot$
If $\exists g' : ((g', u, n, m, ad), c) \in C$ then return $m$
If $\neg\mathsf{user\_is\_corrupt}[u]$ then
  If $T[u, m_s, s] = \bot$ then
    $\mathsf{bad}_0 \leftarrow \mathsf{true}$   // $\mathcal{A}_{\mathsf{SUFCMA}}$ breaks SUFCMA of DS.
    $\mathsf{win} \leftarrow \mathsf{true}$   // $\mathcal{G}_{[0,1)}$
  Else
    If $T[u, m_s, s] \neq m$ then
      $\mathsf{bad}_1 \leftarrow \mathsf{true}$   // $\mathcal{A}_{\mathsf{CR}}$ breaks CR of H.
      $\mathsf{win} \leftarrow \mathsf{true}$   // $\mathcal{G}_{[0,2)}$
    Else   // $T[u, m_s, s] = m$
      $\mathsf{bad}_2 \leftarrow \mathsf{true}$   // Contradicts the tidiness of NE.
      $\mathsf{win} \leftarrow \mathsf{true}$   // $\mathcal{G}_{[0,3)}$
Return $m$
```

```
Adversary $\mathcal{A}_{\mathsf{SUFCMA}}^{\textsc{U},\textsc{Sign}}$
$\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}^{\textsc{SimU},\textsc{SimG},\textsc{SimSigEnc},\textsc{SimVerDec}}$

$\textsc{SimSigEnc}(g, u, n, m, ad)$   // $ad = \varepsilon$
require $\mathsf{K}[g] \neq \bot$ and $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$h \leftarrow \mathsf{H}(m)$ ; $m_s \leftarrow$ "Keybase-Chat-2" $\|$ $\langle \mathsf{K}[g], n, h \rangle$
$s \leftarrow \textsc{Sign}(u, m_s)$ ; $T[u, m_s, s] \leftarrow m$
$m_e \leftarrow s \,\|\, m$ ; $c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_e)$
$C \leftarrow C \cup \{((g, u, n, m, ad), c)\}$ ; Return $c$

$\textsc{SimNewHonGroup}(g, \mathsf{users})$
$\textsc{SimExposeGroup}(g)$
$\textsc{SimNewCorrGroup}(g, K, \mathsf{users})$
// These oracles are identical to the corresponding
// oracles in the IUF game for SP as per Fig. 7.
```

```
$\textsc{SimVerDec}(g, u, n, c, ad)$   // $ad = \varepsilon$
require $\mathsf{K}[g] \neq \bot$ and $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$m_e \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c)$
If $m_e = \bot$ then return $\bot$
$s \,\|\, m \leftarrow m_e$   // s.t. $|s| = \mathsf{DS.sl}$, $|m| \geq 0$
$h \leftarrow \mathsf{H}(m)$ ; $m_s \leftarrow$ "Keybase-Chat-2" $\|$ $\langle \mathsf{K}[g], n, h \rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], m_s, s)$ then return $\bot$
If $\exists g' : ((g', u, n, m, ad), c) \in C$ then return $m$
If $\neg\mathsf{user\_is\_corrupt}[u]$ then
  If $T[u, m_s, s] = \bot$ then
    $z \leftarrow (u, m_s, s)$
    $\mathbf{abort}(z)$
Return $m$
```

```
$\textsc{SimNewHonUser}(u)$
require $\mathsf{sk}[u] = \mathsf{vk}[u] = \bot$
$\mathsf{sk}[u] \leftarrow$ "dummy"
$\mathsf{vk}[u] \leftarrow \textsc{NewHonUser}(u)$
Return $\mathsf{vk}[u]$
```

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

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

Fig. 23: **Top pane:** Games $\mathcal{G}_0$–$\mathcal{G}_3$ for the proof of Theorem 3. The code highlighted in gray was added by expanding the symmetric signcryption algorithms $\mathsf{SP.SigEnc}, \mathsf{SP.VerDec}$ and the ciphertext-triviality predicate $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}group}}$ in game $\mathcal{G}_{\mathsf{SP},\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}group}}}^{\mathsf{IUF}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}})$. The code added for the transitions between games is highlighted in green. **Bottom pane:** Adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ for the proof of Theorem 3. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_0$.

By combining the above, we have

$$\begin{aligned}
\mathsf{Adv}_{\mathsf{SP},\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}group}}}^{\mathsf{IUF}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}) = \Pr[\mathcal{G}_0] &= \sum_{i=0}^{2}(\Pr[\mathcal{G}_i] - \Pr[\mathcal{G}_{i+1}]) + \Pr[\mathcal{G}_3] \\
&\leq \sum_{i=0}^{2} \Pr[\mathsf{bad}_i^{\mathcal{G}_i}] + \Pr[\mathcal{G}_3] \\
&\leq \mathsf{Adv}_{\mathsf{DS}}^{\mathsf{SUFCMA}}(\mathcal{A}_{\mathsf{SUFCMA}}) + \mathsf{Adv}_{\mathsf{H}}^{\mathsf{CR}}(\mathcal{A}_{\mathsf{CR}}).
\end{aligned}$$

We justify our claims in the following paragraphs.

Game $\mathcal{G}_0$. Game $\mathcal{G}_0$ is functionally equivalent to game $\mathcal{G}^{\mathsf{IUF}}_{\mathsf{SP},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}}$. The former expands multiple instructions in the latter; the expanded code is marked in gray. Game $\mathcal{G}_0$ also contains newly added code that is highlighted in green and does not affect its functionality; this code will be used for transitions between games. In particular, the `if-then-else` statement that was added in oracle VerDec sets the win flag in each of its conditional branches. It follows that

$$\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{SP},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}) = \Pr[\mathcal{G}_0].$$

Transitions from $\mathcal{G}_0$ to $\mathcal{G}_3$. Let $i \in \{0, 1, 2\}$. Games $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ are identical until bad_i is set, therefore

$$\Pr[\mathcal{G}_i] - \Pr[\mathcal{G}_{i+1}] \leq \Pr[\mathsf{bad}_i^{\mathcal{G}_i}].$$

Each of these transitions removes an instruction that sets the win flag. The win flag can no longer be set in game $\mathcal{G}_3$, making it impossible for $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$ to win. We have

$$\Pr[\mathcal{G}_3] = 0.$$

Bounding $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$. In Fig. 23 we build an adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ against the SUFCMA security of DS. It perfectly simulates game $\mathcal{G}_0$ for adversary $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$ until the bad_0 flag is set. Adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ responds to $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$'s user-oracle queries using its own user oracles, and it simulates group oracles by sampling and maintaining group keys locally. While simulating the SigEnc oracle, it substitutes calls to the signing algorithm $\mathsf{DS.Sig}$ with its signing oracle Sign. Whenever bad_0 is set, adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ halts the simulation and returns (u, m_s, s) as the forgery in its own game. Note that in any VerDec query that sets bad_0, the following conditions must be satisfied: $\mathsf{DS.Ver}(\mathsf{vk}[u], m_s, s)$, $\neg\mathsf{user\_is\_corrupt}[u]$, and $T[u, m_s, s] = \perp$. These are equivalent to the three conditions that are necessary to win in the unforgeability game against DS. We have

$$\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}}).$$

Bounding $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$. In order to upper-bound $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$, consider an adversary $\mathcal{A}_{\mathsf{CR}}$ against the collision resistance of H that is defined as follows. Adversary $\mathcal{A}_{\mathsf{CR}}$ perfectly simulates game $\mathcal{G}_1$ for $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$ until the bad_1 flag is set. To simulate the user oracles U and the group oracles G, adversary $\mathcal{A}_{\mathsf{CR}}$ itself samples and maintains all user keys and group keys. Whenever the bad_1 flag is set in VerDec, adversary $\mathcal{A}_{\mathsf{CR}}$ returns a pair of strings $(T[u, m_s, s], m)$. Note that immediately before setting bad_1, game $\mathcal{G}_1$ verifies that the table entry $T[u, m_s, s]$ is not empty and that the two strings are distinct. We now show that these strings also produce the same hash value. First, observe that the table T is filled by oracle SigEnc. For any string assigned to the table entry $T[u, m_s, s]$, we know that $h = \mathsf{H}(T[u, m_s, s])$ is true for the hash value h that is encoded in m_s. But while reconstructing the value of m_s in the current VerDec call, the same hash must have been obtained upon evaluating $h \leftarrow \mathsf{H}(m)$. So the strings returned by $\mathcal{A}_{\mathsf{CR}}$ produce a collision. It follows that

$$\Pr[\mathsf{bad}_1^{\mathcal{G}_1}] \leq \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{CR}}).$$

Bounding $\Pr[\mathsf{bad}_2^{\mathcal{G}_2}]$. Consider a VerDec query $(g, u, n, c, \varepsilon)$ that sets bad_2 in game $\mathcal{G}_2$. Immediately prior to setting bad_2, game $\mathcal{G}_2$ verifies that $T[u, m_s, s] = m$ for $m_s =$ "Keybase-Chat-2" $\| \langle \mathsf{K}[g], n, h \rangle$ that was obtained during the current VerDec call. Since the table T is only populated inside SigEnc, it follows that the SigEnc oracle was previously queried with $(g', u, n, m, \varepsilon)$ as input, for some g' satisfying $\mathsf{K}[g'] = \mathsf{K}[g]$. In that SigEnc query, let the output of $\mathsf{NE.Enc}$ be c'. Then $((g', u, n, m, \varepsilon), c') \in C$. However, the VerDec oracle in game $\mathcal{G}_2$ also requires that $\nexists g' \colon ((g', u, n, m, \varepsilon), c) \in C$ before the bad_2 flag could be set. This is only possible if adversary $\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}}$ was able to find a distinct NE ciphertext $c \neq c'$ that decrypts to the same value as c'. By combining the information we deduced about the SigEnc and VerDec queries above, we have $\mathsf{NE.Enc}(\mathsf{K}[g], n, \mathsf{NE.Dec}(\mathsf{K}[g], n, c, \varepsilon), \varepsilon) \neq c'$. This contradicts the assumed tidiness property of NE, therefore

$$\Pr[\mathsf{bad}_2^{\mathcal{G}_2}] = 0.$$

C Analysis and Proofs for the Out-Group AE Security of Keybase

C.1 $\mathcal{M}$-sparsity of Ed25519

In Section 6.3 we defined the $\mathcal{M}$-SPARSE security of a digital signature scheme for an arbitrary set $\mathcal{M} \subseteq \{0,1\}^*$. In this section, we reduce the $\mathcal{M}$-SPARSE security of Ed25519 for any sufficiently small $\mathcal{M}$. To do so, we first introduce and analyze two definitions. The first is Discrete Logarithm with Prefix Matching (DL-PM), which is a variant of the Discrete Logarithm (DL) assumption. We verify the DL-PM assumption in the generic group model. The second is Impersonation with Prefix Matching (IMP-PM), which is a new security definition for identification schemes. Intuitively, IMP-PM captures an adversary's inability to generate a conversation transcript in which the commitment contains a predefined prefix such that an honest verifier would accept the transcript even if the adversary is allowed to choose the verification key used by the verifier. We prove that the IMP-PM security of the identification scheme underlying Ed25519 is implied by the hardness of the DL-PM assumption in the underlying prime-order group. We then use this result to prove that Ed25519 achieves SPARSE security. We introduce these definitions with a particular focus on their relevance to our analysis of Keybase. Our primary intention is not to present these definitions as stand-alone concepts of independent interest but rather to use them as tools essential for our examination of Keybase's SEAL-PACKET-SS scheme.

NOTATION AND CONVENTIONS. For any positive integer $n \in \mathbb{N} \setminus \{0\}$, we let $[n] = \{1, 2, \ldots, n\}$. We use $\mathbb{G}$ to denote a group. We use the additive notation to write group operations and use $0_{\mathbb{G}}$ to denote the identity element of $\mathbb{G}$. We assume that the membership test is efficiently computable in $\mathbb{G}$ for all groups we consider. Wherever specified, we let the elements of $\mathbb{G}$ be represented as k-bit strings where k is a positive integer. For a cyclic group $\mathbb{G}_p$ of order p with a generator $\mathbf{B}$, the discrete logarithm of an element $X \in \mathbb{G}_p$ is the value $x \in \mathbb{Z}_p$ such that $x\mathbf{B} = X$.

```
Game $\mathcal{G}^{\mathsf{DL\text{-}PM}}_{\mathbb{G}_p,\mathcal{M}}(\mathcal{A}_{\mathsf{DL\text{-}PM}})$
$(r, R, \gamma) \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{DL\text{-}PM}}$
$\mathsf{win}_0 \leftarrow (R = r\mathbf{B})$
$\mathsf{win}_1 \leftarrow (R \,\|\, \gamma \in \mathcal{M})$
Return $\mathsf{win}_0$ and $\mathsf{win}_1$
```

```
Game $\mathcal{G}^{\mathsf{IMP\text{-}PM}}_{\mathsf{ID},\mathcal{M}}(\mathcal{A}_{\mathsf{IMP\text{-}PM}})$
$(vk, \mathsf{com}, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{IMP\text{-}PM}}$
$\mathsf{ch} \leftarrow\!\!\$\ \mathsf{ID.V}_1$
$(\mathsf{rsp}, \gamma) \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{IMP\text{-}PM}}(\mathsf{ch}, \mathsf{st})$
$\mathsf{win}_0 \leftarrow \mathsf{ID.V}_2(vk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp})$
$\mathsf{win}_1 \leftarrow (\mathsf{com} \,\|\, \gamma \in \mathcal{M})$
Return $\mathsf{win}_0$ and $\mathsf{win}_1$
```

Fig. 24: **Left pane:** Game defining the discrete logarithm with prefix matching security of a group $\mathbb{G}_p$ with respect to a message space $\mathcal{M}$. **Right pane:** Game defining the impersonation with prefix matching security of an identification scheme ID with respect to a message space $\mathcal{M}$.

THE DL-PM ASSUMPTION FOR $\mathbb{G}_p$. Let $\mathbb{G}_p$ be a cyclic group of order p, and let $\mathbf{B} \in \mathbb{G}_p$ be its generator. Consider game $\mathcal{G}^{\mathsf{DL\text{-}PM}}$ of Fig. 24, defined for $\mathbb{G}_p$, a set $\mathcal{M} \subseteq \{0,1\}^*$, and an adversary $\mathcal{A}_{\mathsf{DL\text{-}PM}}$. The advantage of $\mathcal{A}_{\mathsf{DL\text{-}PM}}$ against the DL-PM security of $\mathbb{G}_p$ with respect to $\mathcal{M}$ is defined as $\mathsf{Adv}^{\mathsf{DL\text{-}PM}}_{\mathbb{G}_p,\mathcal{M}}(\mathcal{A}_{\mathsf{DL\text{-}PM}}) = \Pr[\mathcal{G}^{\mathsf{DL\text{-}PM}}_{\mathbb{G}_p,\mathcal{M}}(\mathcal{A}_{\mathsf{DL\text{-}PM}})]$. The adversary wins if it finds a tuple (r, R, γ) such that r is the discrete logarithm of R and $R \,\|\, \gamma \in \mathcal{M}$. We prove the hardness of DL-PM in the generic group model for sufficiently small $\mathcal{M}$.

IDENTIFICATION SCHEMES. An identification scheme ID is an interactive protocol involving two parties, the prover and the verifier. The prover is instantiated with a secret key sk that it uses to send the first message called a commitment. The verifier, instantiated with a public key vk, then samples and sends a random challenge. Lastly, the prover sends a response to the verifier's challenge. The verifier uses vk and the conversation transcript to accept or reject the conversation. Formally, ID specifies three algorithms ID.Kg, $\mathsf{ID.P} = (\mathsf{P}_1, \mathsf{P}_2)$, and $\mathsf{ID.V} = (\mathsf{V}_1, \mathsf{V}_2)$. Associated to every instance of ID is a challenge set ChalSet. The key generation algorithm ID.Kg returns a secret key sk and a public verification key vk.

The prover $\mathsf{ID.P}$ is a pair of algorithms $\mathsf{ID.P}_1$ and $\mathsf{ID.P}_2$. The commitment algorithm $\mathsf{ID.P}_1$ takes sk as input and returns a commitment com and some state st. The deterministic response algorithm $\mathsf{ID.P}_2$ takes a challenge $\mathsf{ch} \in \mathsf{ChalSet}$ sent by the verifier $\mathsf{ID.V}_1$ and a state st, and returns a response rsp to send to the verifier $\mathsf{ID.V}_2$. The verifier $\mathsf{ID.V}$ is a pair of algorithms $\mathsf{ID.V}_1$ and $\mathsf{ID.V}_2$. The algorithm $\mathsf{ID.V}_1$ samples and returns a random challenge $\mathsf{ch} \in \mathsf{ChalSet}$. The deterministic verification algorithm $\mathsf{ID.V}_2$ takes the tuple $(vk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp})$ as input and returns true if it accepts the conversation, and false otherwise.

<u>The IMP-PM Security of ID.</u> The IMP-PM game $\mathcal{G}^{\mathsf{IMP\text{-}PM}}_{\mathsf{ID},\mathcal{M}}$ for an identification scheme ID, a message space $\mathcal{M}$, and an adversary $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ is shown in Fig. 24. The advantage of $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ against the $\mathsf{IMP\text{-}PM}$ security of ID is given by $\mathsf{Adv}^{\mathsf{IMP\text{-}PM}}_{\mathsf{ID},\mathcal{M}}(\mathcal{A}_{\mathsf{IMP\text{-}PM}}) = \Pr[\mathcal{G}^{\mathsf{IMP\text{-}PM}}_{\mathsf{ID},\mathcal{M}}(\mathcal{A}_{\mathsf{IMP\text{-}PM}})]$. The adversary $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ is run twice over the course of the game. Its first execution is at the beginning of the game after which it returns the tuple $(vk, \mathsf{com}, \mathsf{st})$. The game then proceeds to sample the challenge ch using $\mathsf{ID.V}_1$. Subsequently, the adversary is run once again on input $(\mathsf{ch}, \mathsf{st})$. Its final output is (rsp, γ). The adversary wins if $\mathsf{ID.V}_2(vk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp}) = \mathsf{true}$ and $\mathsf{com} \,\|\, \gamma \in \mathcal{M}$.

<u>SPARSE Security of the Ed25519 Signature Scheme.</u>

Construction 6. *Let $k = 256$ and prime $p \approx 2^{252}$. Let $\mathbb{G}$ be a group whose elements are encoded as k-bit strings. Let $\mathbb{G}_p$ be the cyclic subgroup of $\mathbb{G}$ whose order is p and generator is $\mathbf{B}$. Let $\mathsf{H}_1 = \mathsf{H}_2 = \mathsf{H}_3 =$ $\mathsf{SHA\text{-}512}$. Then $\mathsf{Ed25519\text{-}DS}[k, p, \mathbb{G}, \mathbb{G}_p, \mathbf{B}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3]$ is the signature scheme as defined in Fig. 25 with $\mathsf{Ed25519\text{-}DS.skl} = 256$ and $\mathsf{Ed25519\text{-}DS.sl} = 512$.*

```
Ed25519.Kg
$sk \leftarrow\!\!\$\ \{0,1\}^k$                       // $k = 256$
$e_1 \,\|\, e_2 \leftarrow \mathsf{H}_1(sk)$           // $\mathsf{H}_1 = \mathsf{SHA\text{-}512}$
                                                       // $|e_1| = |e_2| = 256$
$t \leftarrow 2^{k-2} + \sum_{i=3}^{k-3} 2^i \cdot e_1[i]$
$a \leftarrow t \bmod p$                               // $p \approx 2^{252}$
$vk \leftarrow a\mathbf{B}$
Return $(sk, vk)$
```

```
Ed25519.Sig$(sk, m)$
$e_1 \,\|\, e_2 \leftarrow \mathsf{H}_1(sk)$
$t \leftarrow 2^{k-2} + \sum_{i=3}^{k-3} 2^i \cdot e_1[i]$
$a \leftarrow t \bmod p$
$r \leftarrow \mathsf{H}_2(e_2, m)$                    // $\mathsf{H}_2 = \mathsf{SHA\text{-}512}$
$R \leftarrow r\mathbf{B}$ ; $A \leftarrow a\mathbf{B}$
$\mathsf{ch} \leftarrow \mathsf{H}_3(R, A, m)$         // $\mathsf{H}_3 = \mathsf{SHA\text{-}512}$
$z \leftarrow (r + \mathsf{ch} \cdot a) \bmod p$
$s \leftarrow R \,\|\, z$
Return $s$
```

```
Ed25519.Ver$(vk, m, s)$
$R \,\|\, z \leftarrow s$                              // $|R| = |z| = 256$
$A \leftarrow vk$
If $z \geq p$ then return false
If $R \notin \mathbb{G}_p$ or $A \notin \mathbb{G}$ then return false
$\mathsf{ch} \leftarrow \mathsf{H}_3(R, A, m)$
Return $z \cdot \mathbf{B} = R + \mathsf{ch} \cdot A$
```

Fig. 25: The $\mathsf{Ed25519}$ signature scheme.

Proposition 5 **($\mathsf{IMP\text{-}PM}_{\mathsf{EdID}} \Rightarrow \mathsf{SPARSE}_{\mathsf{Ed25519}}$).** *Let $\mathsf{Ed25519} = \mathsf{Ed25519\text{-}DS}[k, p, \mathbb{G}, \mathbb{G}_p, \mathbf{B}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3]$ be the digital signature scheme built from k, p, $\mathbb{G}$, $\mathbb{G}_p$, $\mathbf{B}$, H_1, H_2, H_3 as specified in Construction 6. Let $\mathcal{M}$ be a message space that does not depend on H_3. Let $\mathcal{A}_{\mathsf{SPARSE}}$ be any adversary against the $\mathcal{M}$-SPARSE security of $\mathsf{Ed25519}$. Let H_3 be modeled as a random oracle and q_{H} be the number of random oracle queries that $\mathcal{A}_{\mathsf{SPARSE}}$ makes. Then we build an adversary $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ such that*

$$\mathsf{Adv}^{\mathsf{SPARSE}}_{\mathsf{Ed25519},\mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}}) \leq (q_{\mathrm{H}} + 1) \cdot \mathsf{Adv}^{\mathsf{IMP\text{-}PM}}_{\mathsf{EdID},\mathcal{M}}(\mathcal{A}_{\mathsf{IMP\text{-}PM}}).$$

Proof. We begin with the game $\mathcal{G}^{\mathsf{SPARSE}}_{\mathsf{Ed25519},\mathcal{M}}$ in Fig. 26. The hash function H_3 in Fig. 27 is implicitly used as $\mathsf{H}_3(\cdot) \bmod p$. So we use the indifferentiability results of Bellare, Davis, and Di [BDD23] to model H_3 as a random oracle. We have

$$\Pr[\mathcal{G}^{\mathsf{SPARSE}}_{\mathsf{Ed25519},\mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}})] = \mathsf{Adv}^{\mathsf{SPARSE}}_{\mathsf{Ed25519},\mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}}).$$

In Fig. 26 we build an adversary $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ against the $\mathsf{IMP\text{-}PM}$ security of EdID that perfectly simulates the game $\mathcal{G}^{\mathsf{SPARSE}}_{\mathsf{Ed25519},\mathcal{M}}$ for $\mathcal{A}_{\mathsf{SPARSE}}$. The adversary $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ guesses the random oracle query i for which $\mathcal{A}_{\mathsf{SPARSE}}$ will ultimately attempt a forgery. Note that $\mathcal{A}_{\mathsf{SPARSE}}$ may attempt a forgery on values that it never queried to the random oracle. So $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ makes a call to the random oracle after $\mathcal{A}_{\mathsf{SPARSE}}$ has attempted to forge a signature. Therefore $i \in [q_{\mathrm{H}} + 1]$. For every query $j \in [q_{\mathrm{H}} + 1]$ such that $j \neq i$,

```
$\mathcal{G}^{\mathsf{SPARSE}}_{\mathsf{Ed25519},\mathcal{M}}$
$(vk, m, s, \gamma) \leftarrow\$ \mathcal{A}^{\mathrm{H}}_{\mathsf{SPARSE}}$
$R \,\|\, z \leftarrow s$ ; $A \leftarrow vk$
If $z \geq p$ then return $\perp$
If $R \notin \mathbb{G}_p$ or $A \notin \mathbb{G}$ then return $\mathsf{false}$
$\mathsf{ch} \leftarrow \mathrm{H}(R, A, m)$
If $z \cdot \mathbf{B} = R + \mathsf{ch} \cdot A$ then
    $\mathsf{win}_0 \leftarrow \mathsf{true}$
$\mathsf{win}_1 \leftarrow (s \,\|\, \gamma \in \mathcal{M})$
Return $\mathsf{win}_0$ and $\mathsf{win}_1$
```

```
$\mathrm{H}(x)$
If $\mathsf{H}[x] = \perp$ then
    $X \leftarrow\$ \mathbb{Z}_p$
    $\mathsf{H}[x] \leftarrow X$
Return $\mathsf{H}[x]$
```

```
$\mathcal{A}_{\mathsf{IMP\text{-}PM}}$
$i \leftarrow\$ [q_{\mathrm{H}} + 1]$ ; $j \leftarrow 0$
$(vk, m, s, \gamma) \leftarrow\$ \mathcal{A}^{\mathrm{SIMH}}_{\mathsf{SPARSE}}$
$(R \,\|\, z) \leftarrow s$
$x \leftarrow (R, vk, m)$
$\mathrm{SIMH}(x)$
Return $(R, z \,\|\, \gamma)$
```

```
$\mathrm{SIMH}(x)$
If $\mathsf{H}[x] \neq \perp$ then return $\mathsf{H}[x]$
$j = j + 1$
If $j = i$ then
    $(R, A, m) \leftarrow x$
    $R^* \leftarrow R$ ; $A^* \leftarrow A$
    $\mathsf{ch}^* \leftarrow \mathbf{output}(A^*, R^*)$
    $\mathsf{H}[x] \leftarrow \mathsf{ch}^*$
    Return $\mathsf{ch}^*$
$X \leftarrow\$ \mathbb{Z}_p$
$\mathsf{H}[x] \leftarrow X$
Return $\mathsf{H}[x]$
```

Fig. 26: **Left pane:** The game used in the proof of Proposition 5. **Right pane:** The adversary $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ used in the proof of Proposition 5. The function **output** is described in the text below.

$\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ lazily simulates the output of H. For the $j = i$ case, we introduce a new instruction **output** which is to be interpreted as follows. When $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ encounters this instruction during the i^{th} random oracle query (R^*, A^*, m^*), it returns the values A^*, R^*, and its current state st to its own game. It is then run on the inputs $(\mathsf{ch}, \mathsf{st})$. It uses the state st to resume executing in the context of SIMH, assigning ch to the variable ch^*. Finally $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ returns ch as the output of the i^{th} H query. Recall that the challenge ch in $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$'s game is sampled uniformly at random from $\mathbb{Z}_p$. Also, note that **output** is guaranteed to execute exactly once. So, this change is undetectable for $\mathcal{A}_{\mathsf{SPARSE}}$.

At some point, $\mathcal{A}_{\mathsf{SPARSE}}$ returns the forgery (vk, m, s, γ) where s is of the form $R \,\|\, z$. Adversary $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ returns $(R, z \,\|\, \gamma)$ and wins whenever $(vk, R) = (A^*, R^*)$ and $\mathcal{A}_{\mathsf{SPARSE}}$ wins. We know that

$$\Pr[(vk, R) = (A^*, R^*)] \geq 1/(q_{\mathrm{H}} + 1)$$

Consider the winning conditions for $\mathcal{A}_{\mathsf{SPARSE}}$. We have $A^* \in \mathbb{G}$, $R^* \in \mathbb{G}_p$, $z \cdot \mathbf{B} = R^* + \mathsf{ch}^* \cdot A^*$ and $s \,\|\, \gamma \in \mathcal{M}$ all hold. The last condition simplifies to $R \,\|\, (z \,\|\, \gamma) \in \mathcal{M}$. These are exactly the same as the winning conditions for $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$. Hence,

$$\mathsf{Adv}^{\mathsf{IMP\text{-}PM}}_{\mathsf{EdID},\mathcal{M}}(\mathcal{A}_{\mathsf{IMP\text{-}PM}}) \geq 1/(q_{\mathrm{H}} + 1) \cdot \mathsf{Adv}^{\mathsf{SPARSE}}_{\mathsf{Ed25519},\mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}})$$

This concludes the proof.

IMP-PM Security of the ID Scheme Underlying Ed25519.

Construction 7. *Let $k = 256$ and prime $p \approx 2^{252}$. Let $\mathbb{G}$ be a group whose elements are encoded as k-bit strings. Let $\mathbb{G}_p$ be the cyclic subgroup of $\mathbb{G}$ whose order is p and generator is $\mathbf{B}$. Let $\mathsf{H} = \mathsf{SHA\text{-}512}$. Then $\mathsf{Ed25519\text{-}ID}[k, p, \mathbb{G}, \mathbb{G}_p, \mathbf{B}, \mathsf{H}]$ is the identification scheme defined in Fig. 27 with $\mathsf{Ed25519\text{-}ID.ChalSet} = \mathbb{Z}_p$.*

Proposition 6 ($\mathsf{DL\text{-}PM}_{\mathbb{G}_p} \Rightarrow \mathsf{IMP\text{-}PM}_{\mathsf{EdID}}$). *Let $\mathsf{EdID} = \mathsf{Ed25519\text{-}ID}[k, p, \mathbb{G}, \mathbb{G}_p, \mathbf{B}, \mathsf{H}]$ be the identification protocol built from k, p, $\mathbb{G}$, $\mathbb{G}_p$, $\mathbf{B}$, H as specified in Construction 7. Let $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ be any adversary against the $\mathsf{IMP\text{-}PM}$ security of EdID with respect to the message space $\mathcal{M}$. Then, we build an adversary $\mathcal{A}_{\mathsf{DL\text{-}PM}}$ such that*

$$\mathsf{Adv}^{\mathsf{IMP\text{-}PM}}_{\mathsf{EdID},\mathcal{M}}(\mathcal{A}_{\mathsf{IMP\text{-}PM}}) \leq \sqrt{\mathsf{Adv}^{\mathsf{DL\text{-}PM}}_{\mathbb{G}_p,\mathcal{M}}(\mathcal{A}_{\mathsf{DL\text{-}PM}})} + 1/p.$$

Proof. We start with the game $\mathcal{G}^{\mathsf{IMP\text{-}PM}}_{\mathsf{EdID},\mathcal{M}}$ in Fig. 28. We have

$$\Pr[\mathcal{G}(\mathcal{A}_{\mathsf{IMP\text{-}PM}})] = \mathsf{Adv}^{\mathsf{IMP\text{-}PM}}_{\mathsf{EdID},\mathcal{M}}(\mathcal{A}_{\mathsf{IMP\text{-}PM}}).$$

Adversary $\mathcal{A}_{\mathsf{DL\text{-}PM}}$ runs $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ and simulates the game $\mathcal{G}^{\mathsf{IMP\text{-}PM}}_{\mathsf{EdID},\mathcal{M}}$. At some point, $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ returns the tuple $(vk, \mathsf{com}, \mathsf{st})$. Then $\mathcal{A}_{\mathsf{DL\text{-}PM}}$ samples a challenge ch_0 uniformly at random from $\mathbb{Z}_p$ and runs $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ on the inputs $(\mathsf{ch}_0, \mathsf{st})$. Finally, $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ terminates with output (z_0, γ_0). Subsequently, $\mathcal{A}_{\mathsf{DL\text{-}PM}}$ resets

```
$\underline{\mathsf{EdID.Kg}}$
$sk \leftarrow\$ \{0,1\}^k$          // $k = 256$
$e_1 \| e_2 \leftarrow \mathsf{H}(sk)$        // $\mathsf{H} = \mathsf{SHA\text{-}512}$
                                // $|e_1| = |e_2| = 256$
$t \leftarrow 2^{k-2} + \sum_{i=3}^{k-3} 2^i \cdot e_1[i]$
$a \leftarrow t \bmod p$           // $p \approx 2^{252}$
$vk \leftarrow a\mathbf{B}$
Return $(sk, vk)$
```

```
$\underline{\mathsf{EdID.P}_1(sk)}$
$e_1 \| e_2 \leftarrow \mathsf{H}(sk)$
$t \leftarrow 2^{k-2} + \sum_{i=3}^{k-3} 2^i \cdot e_1[i]$
$a \leftarrow t \bmod p$ ; $r \leftarrow\$ \mathbb{Z}_p$
$\mathsf{st} \leftarrow (r, a)$
$R \leftarrow r\mathbf{B}$ ; $\mathsf{com} \leftarrow R$
Return $(\mathsf{com}, \mathsf{st})$

$\underline{\mathsf{EdID.V}_1()}$
$\mathsf{ch} \leftarrow\$ \mathbb{Z}_p$
Return $\mathsf{ch}$
```

```
$\underline{\mathsf{EdID.P}_2(\mathsf{ch}, \mathsf{st})}$
$(r, a) \leftarrow \mathsf{st}$
$z \leftarrow (r + \mathsf{ch} \cdot a) \bmod p$
$\mathsf{rsp} \leftarrow z$ ;  Return $\mathsf{rsp}$

$\underline{\mathsf{EdID.V}_2(vk, \mathsf{com}, \mathsf{ch}, \mathsf{rsp})}$
If $\mathsf{com} \notin \mathbb{G}_p$ or $vk \notin \mathbb{G}$ then return $\mathsf{false}$
$R \leftarrow \mathsf{com}$ ; $A \leftarrow vk$ ; $z \leftarrow \mathsf{rsp}$
Return $z \cdot \mathbf{B} = R + \mathsf{ch} \cdot A$
```

Fig. 27: The identification protocol EdID for the proof of Proposition 6.

```
$\underline{\mathcal{G}^{\mathsf{IMP\text{-}PM}}_{\mathsf{EdID},\mathcal{M}}}$
$(vk, \mathsf{com}, \mathsf{st}) \leftarrow\$ \mathcal{A}_{\mathsf{IMP\text{-}PM}}$
$\mathsf{ch} \leftarrow\$ \mathbb{Z}_p$
$(\mathsf{rsp}, \gamma) \leftarrow\$ \mathcal{A}_{\mathsf{IMP\text{-}PM}}(\mathsf{ch}, \mathsf{st})$
If $\mathsf{com} \notin \mathbb{G}_p$ or $vk \notin \mathbb{G}$ then return $\mathsf{false}$
$R \leftarrow \mathsf{com}$ ; $A \leftarrow vk$ ; $z \leftarrow \mathsf{rsp}$
Return $z \cdot \mathbf{B} = R + \mathsf{ch} \cdot A$ and $R \| \gamma \in \mathcal{M}$
```

```
$\underline{\text{Adversary } \mathcal{A}_{\mathsf{DL\text{-}PM}}}$
$(vk, \mathsf{com}, \mathsf{st}) \leftarrow\$ \mathcal{A}_{\mathsf{IMP\text{-}PM}}$
If $\mathsf{com} \notin \mathbb{G}_p$ or $vk \notin \mathbb{G}$ then
    Return $(1, \mathbf{B}, \varepsilon)$
$\mathsf{ch}_0 \leftarrow\$ \mathbb{Z}_p$
$(z_0, \gamma_0) \leftarrow\$ \mathcal{A}_{\mathsf{IMP\text{-}PM}}(\mathsf{ch}_0, \mathsf{st})$
$\mathsf{ch}_1 \leftarrow\$ \mathbb{Z}_p$
$(z_1, \gamma_1) \leftarrow\$ \mathcal{A}_{\mathsf{IMP\text{-}PM}}(\mathsf{ch}_1, \mathsf{st})$
$(Q + T) \leftarrow vk$    // $Q \in \mathbb{G}_p$ ; $T \in \mathbb{G}_t$
$R \leftarrow \mathsf{com}$
If $\mathsf{ch}_0 \neq \mathsf{ch}_1$ then
    If $z_0 \cdot \mathbf{B} = R + \mathsf{ch}_0 \cdot Q$ and $z_1 \cdot \mathbf{B} = R + \mathsf{ch}_1 \cdot Q$ then
        If $R \| \gamma_0 \in \mathcal{M}$ and $R \| \gamma_1 \in \mathcal{M}$ then
            $r \leftarrow z_0 - \mathsf{ch}_0 \cdot (z_0 - z_1)(\mathsf{ch}_0 - \mathsf{ch}_1)^{-1} \bmod p$
            Return $(r, R, \gamma_0)$
Return $(1, \mathbf{B}, \varepsilon)$
```

Fig. 28: **Left pane:** The security game used in the proof of Proposition 6. We use gray highlighting to show code from EdID. **Right pane:** Adversary $\mathcal{A}_{\mathsf{DL\text{-}PM}}$ used in the proof of Proposition 6. The interpretation of the line highlighted green is explained in the text below.

$\mathcal{A}_{\mathsf{IMP\text{-}PM}}$'s state back to the point just after when it had returned $(vk, \mathsf{com}, \mathsf{st})$, chooses a new random challenge ch_1 and sends it to $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$. The adversary $\mathcal{A}_{\mathsf{IMP\text{-}PM}}$ returns another response (z_1, γ_1).

To explain the step $(Q + T) \leftarrow vk$, we present some details about the group $\mathbb{G}$ underlying EdID. The group $\mathbb{G}$ satisfies $n = 8 \cdot p = 2^3 \cdot p$ where n is the order of $\mathbb{G}$. Using Kronecker's decomposition theorem we have that in addition to the prime order subgroup $\mathbb{G}_p$, the group $\mathbb{G}$ contains another subgroup $\mathbb{G}_t$ of order 8 such that the group $\mathbb{G}$ is the direct product of $\mathbb{G}_p$ and $\mathbb{G}_t$, i.e., $\mathbb{G} = \mathbb{G}_p \times \mathbb{G}_t$. This means that every element $X \in \mathbb{G}$ can be uniquely written as $Q + T$ for $Q \in \mathbb{G}_p$ and $T \in \mathbb{G}_t$. We call $\mathbb{G}_t$ the torsion subgroup of $\mathbb{G}$, and T the torsion component of X. Consider the check $z \cdot \mathbf{B} = R + \mathsf{ch} \cdot A$. This is equivalent to $z \cdot \mathbf{B} - R = \mathsf{ch} \cdot A$ which is in turn equivalent to $z \cdot \mathbf{B} - R = \mathsf{ch}(Q + T)$. By the time this check is performed by $\mathcal{A}_{\mathsf{DL\text{-}PM}}$, we know that $z \cdot \mathbf{B} - R \in \mathbb{G}_p$. For equality to hold, $\mathsf{ch}(Q + T) \in \mathbb{G}_p$ must be true. This can only happen if $T = 0_{\mathbb{G}}$. Therefore, $z \cdot \mathbf{B} - R = \mathsf{ch} \cdot A \implies z \cdot \mathbf{B} - R = \mathsf{ch} \cdot Q$.

Let E be the event that all three conditional statements in Fig. 28 evaluate to true i.e. $\mathsf{ch}_0 \neq \mathsf{ch}_1$, $z_0 \cdot \mathbf{B} = R + \mathsf{ch}_0 \cdot Q$, $z_1 \cdot \mathbf{B} = R + \mathsf{ch}_1 \cdot Q$, and $R \| \gamma_i \in \mathcal{M}$ for $i \in \{0, 1\}$ all hold simultaneously. Whenever the event E occurs, $\mathcal{A}_{\mathsf{DL\text{-}PM}}$ computes the discrete log r of R by solving the system of equations $z_0 = r + \mathsf{ch}_0 \cdot q$ and $z_1 = r + \mathsf{ch}_1 \cdot q$, returns (r, R, γ_0) and wins its game. It follows that

$$\mathsf{Adv}^{\mathsf{DL\text{-}PM}}_{\mathbb{G}_p,\mathcal{M}}(\mathcal{A}_{\mathsf{DL\text{-}PM}}) \geq \Pr[E].$$

We use the Reset Lemma [BP02] to bound $\Pr[E]$. The Reset Lemma upper-bounds the advantage of a dishonest prover ($\mathcal{A}_{\mathsf{IMP\text{-}PM}}$) as a function of the probability that the prover produces two accepting responses in an experiment that resets it as

$$\mathsf{Adv}^{\mathsf{IMP\text{-}PM}}_{\mathsf{EdID},\mathcal{M}}(\mathcal{A}_{\mathsf{IMP\text{-}PM}}) \leq 1/\left|\mathsf{ChalSet}\right| + \sqrt{\Pr[E]}$$

where $\mathsf{ChalSet}$ is the set from which the verifier samples challenges. In our case $\mathsf{ChalSet} = \mathbb{Z}_p$. Combining the probabilities above we get

$$\mathsf{Adv}^{\mathsf{IMP\text{-}PM}}_{\mathsf{EdID},\mathcal{M}}(\mathcal{A}_{\mathsf{IMP\text{-}PM}}) \leq \sqrt{\mathsf{Adv}^{\mathsf{DL\text{-}PM}}_{\mathbb{G}_p,\mathcal{M}}(\mathcal{A}_{\mathsf{DL\text{-}PM}})} + 1/p.$$

This concludes the proof.

Shoup's Generic Group Model. We analyze the $\mathsf{DL\text{-}PM}$ security of $\mathbb{G}_p$ in Shoup's generic group model which uses a random injective map $\sigma\colon \mathbb{Z}_p \to S$ for some set of strings S and thinks of the string $\sigma(x)$ as the representation of the group element $x\mathbf{B}$. It gives the adversary access to two oracles, the generator exponentiation oracle Exp and the group operation oracle Op. On input x, the Exp oracle returns $\sigma(x)$ which is the representation of $x\mathbf{B}$. On input (a, b, X, Y), the Op oracle returns the representation of $aX + bY$ if X and Y are valid string representations of group elements and returns $\perp$ otherwise.

DL-PM in Shoup's Generic Group Model.

Proposition 7. *Let $S = \{0,1\}^k$. Let $\mathcal{M} = \left\{\mathsf{pre} \,\|\, \nu \;\middle|\; \nu \in \{0,1\}^*\right\}$ where pre is the 17-byte prefix defined in Section 6.3. Let $\mathcal{A}_{\mathsf{DL\text{-}PM}}$ be any adversary against the $\mathsf{DL\text{-}PM}$ security of $\mathbb{G}_p$ with respect to $\mathcal{M}$, making at most q_{Exp} and q_{Op} queries to its Exp and Op oracles respectively. Then we show that*

$$\mathsf{Adv}^{\mathsf{DL\text{-}PM}}_{\mathbb{G}_p,\mathcal{M}}(\mathcal{A}_{\mathsf{DL\text{-}PM}}) \leq ((q_{\text{Exp}} + q_{\text{Op}} + 2) \cdot (2^{k-|\mathsf{pre}|} + q_{\text{Exp}} + 2 \cdot q_{\text{Op}} + 2))/2^k + (q_{\text{Exp}} + q_{\text{Op}} + 2)^2/p.$$

Overview. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_4$ in Fig. 29 and Fig. 30. We establish the following claims.

- $\mathsf{Adv}^{\mathsf{DL\text{-}PM}}_{\mathbb{G}_p,\mathsf{pre}}(\mathcal{A}_{\mathsf{DL\text{-}PM}}) = \Pr[\mathcal{G}_0]$
- $\Pr[\mathcal{G}_0] = \Pr[\mathcal{G}_1]$
- $\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_1}]$
- $\Pr[\mathsf{bad}_0^{\mathcal{G}_1}] \leq ((q_{\text{Exp}} + q_{\text{Op}} + 2) \cdot (2^{k-|\mathsf{pre}|} + q_{\text{Exp}} + 2 \cdot q_{\text{Op}} + 2))/2^k$
- $\Pr[\mathcal{G}_2] = \Pr[\mathcal{G}_3]$
- $\Pr[\mathcal{G}_3] - \Pr[\mathcal{G}_4] \leq \Pr[\mathsf{bad}_1^{\mathcal{G}_3}]$
- $\Pr[\mathsf{bad}_1^{\mathcal{G}_3}] \leq (q_{\text{Exp}} + q_{\text{Op}} + 2)^2/p$
- $\Pr[\mathcal{G}_4] = 0$

By combining the above, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{DL\text{-}PM}}_{\mathbb{G}_p,\mathsf{pre}}(\mathcal{A}_{\mathsf{DL\text{-}PM}}) &= \Pr[\mathcal{G}_0] = \Pr[\mathcal{G}_1] = (\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2]) + \Pr[\mathcal{G}_2] \\ &\leq \Pr[\mathsf{bad}_0^{\mathcal{G}_1}] + \Pr[\mathcal{G}_3] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_1}] + (\Pr[\mathcal{G}_3] - \Pr[\mathcal{G}_4]) + \Pr[\mathcal{G}_4] \\ &\leq \Pr[\mathsf{bad}_0^{\mathcal{G}_1}] + \Pr[\mathsf{bad}_1^{\mathcal{G}_4}] \\ &\leq ((q_{\text{Exp}} + q_{\text{Op}}) \cdot (2^{k-|\mathsf{pre}|} + q_{\text{Exp}} + 2 \cdot q_{\text{Op}}))/2^k + (q_{\text{Exp}} + q_{\text{Op}} + 2)^2/p.\end{aligned}$$

We justify our claims in the following paragraphs.

Game $\mathcal{G}_0$. We start with the game $\mathcal{G}_0$ in Fig. 29. The game $\mathcal{G}_0$ is a modified version of the game $\mathcal{G}^{\mathsf{DL\text{-}PM}}_{\mathbb{G}_p,\mathcal{M}}$ shown in Fig. 24. We use the notation $\Phi(\mathbb{Z}_p, S)$ to denote the set of all injections with domain $\mathbb{Z}_p$ and range S. The game $\mathcal{G}_0$ replaces the check $R \,\|\, \gamma \in \mathcal{M}$ of $\mathcal{G}^{\mathsf{DL\text{-}PM}}_{\mathbb{G}_p,\mathcal{M}}$ with the check $R \in \mathcal{M}$. Recall that the length of the prefix pre used to define $\mathcal{M}$ is 136 bits long and that group elements are represented as 256-bit strings. Thus $R \,\|\, \gamma \in \mathcal{M}$ is only possible is $R \in \mathcal{M}$. It follows that

$$\mathsf{Adv}^{\mathsf{DL\text{-}PM}}_{\mathbb{G}_p,\mathcal{M}}(\mathcal{A}^{\mathcal{M}}_{\mathsf{DL\text{-}PM}}) = \Pr[\mathcal{G}_0].$$

Transition from $\mathcal{G}_0$ to $\mathcal{G}_1$. The game $\mathcal{G}_1$ performs a lazy simulation of the random injection σ. To do this the game uses the table T, initially empty, that partially represents σ. Every time the adversary makes an Exp query on a value x, the oracle lazily samples the output and returns it. Now the adversary may query Op on inputs (a, b, X, Y) such that either X or Y is not in $\mathsf{Rng}(T)$. For such queries, the oracle Op needs to probabilistically decide if they are in $\mathsf{Rng}(\sigma)$. Whenever the oracle decides that a point $X \in S$ does not have a preimage, it remembers that decision by maintaining a set S_{reject} of rejected points. The oracle assigns a preimage to X with probability $(p - \mathsf{Rng}(T))/(|S| - |\mathsf{Rng}(T)| - |S_{\mathsf{reject}}|)$ by sampling $j \leftarrow\!\!\$\; [|S| - \mathsf{Rng}(T) - |S_{\mathsf{reject}}|]$ and checking if $j \leq p - \mathsf{Rng}(T)$. We compute this probability as follows. There are p points in $\mathsf{Rng}(\sigma)$. This means that when a query (a, b, X, Y) is made to Op such that $X \notin \mathsf{Rng}(T)$ there are at most $p - |\mathsf{Rng}(T)|$ points that the oracle can decide to assign a preimage for.

```
Game $\mathcal{G}_0$
$\sigma \leftarrow\!\!\$\ \Phi(\mathbb{Z}_p, S)$
$\mathbf{B} \leftarrow \textsc{Exp}(1)$
$(r, R) \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{DL\text{-}PM}}^{\textsc{Op},\textsc{Exp}}(\mathbf{B})$
$\mathsf{win}_0 \leftarrow (R = \textsc{Exp}(r))$
$\mathsf{win}_1 \leftarrow (R[1..|\mathsf{pre}|] = \mathsf{pre})$
Return $\mathsf{win}_0$ and $\mathsf{win}_1$
```

```
$\textsc{Exp}(x)$
Return $\sigma(x)$
```

```
$\textsc{Op}(a, b, X, Y)$
If $\sigma^{-1}(X) \neq \perp$ and $\sigma^{-1}(Y) \neq \perp$ then
    Return $\sigma(a \cdot \sigma^{-1}(X) + b \cdot \sigma^{-1}(Y))$
```

Fig. 29: Game $\mathcal{G}_0$ used in the proof of Proposition 7. We use the notation $\Phi(\mathbb{Z}_p, S)$ to denote the set of all injections with domain $\mathbb{Z}_p$ and range S.

The codomain of σ contains $|S|$ points of which $|\mathsf{Rng}(T)|$ already have a preimage and $|S_{\mathsf{reject}}|$ have been rejected. This leaves a total of $|S| - |\mathsf{Rng}(T)| - |S_{\mathsf{reject}}|$ points in the codomain. Therefore the probability that a point $X \notin \mathsf{Rng}(T)$ belongs in $\mathsf{Rng}(\sigma)$ is $(p - \mathsf{Rng}(T))/(|S| - |\mathsf{Rng}(T) - |S_{\mathsf{reject}}||)$. The table T perfectly simulates the injection σ. Therefore,

$$\Pr[\mathcal{G}_0] = \Pr[\mathcal{G}_1].$$

```
Games $\mathcal{G}_{1-4}$
$i \leftarrow 0$
$\mathbf{B} \leftarrow \textsc{Exp}(1)$
$(r, R) \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{DL\text{-}PM}}^{\textsc{Op},\textsc{Exp}}(\mathbf{B})$
$\mathsf{win}_0 \leftarrow (R = \textsc{Exp}(r))$
$\mathsf{win}_1 \leftarrow (R[1..|\mathsf{pre}|] = \mathsf{pre})$
Return $\mathsf{win}_0$ and $\mathsf{win}_1$
```

```
$\textsc{Exp}(x)$
If $T[x] = \perp$ then
    If $\exists \mathcal{P} \in \mathsf{Dom}(T)\colon \mathcal{P}(\boldsymbol{z}) = x(\boldsymbol{z})$ then    // $\mathcal{G}_{[3,\infty)}$
        $\mathsf{bad}_1 \leftarrow \mathsf{true}$                                   // $\mathcal{G}_{[3,\infty)}$
        $T[x] \leftarrow T[\mathcal{P}]$                                    // $\mathcal{G}_{[3,4)}$
        Return $T[x]$                                         // $\mathcal{G}_{[3,4)}$
    $T[x] \leftarrow\!\!\$\ S \setminus (\mathsf{Rng}(T) \cup S_{\mathsf{reject}})$
    If $T[x][1..|\mathsf{pre}|] = \mathsf{pre}$ then
        $\mathsf{bad}_0 \leftarrow \mathsf{true}$
        $T[x] \leftarrow\!\!\$\ S \setminus (\mathsf{Rng}(T) \cup S_{\mathsf{reject}} \cup S_{\mathsf{pre}})$ // $\mathcal{G}_{[2,\infty)}$
Return $T[x]$
```

```
$\textsc{Op}(a, b, X, Y)$
For $Z \in \{X, Y\}$ do
    If $Z \notin (\mathsf{Rng}(T) \cup S_{\mathsf{reject}})$ then
        $j \leftarrow\!\!\$\ [|S| - \mathsf{Rng}(T) - |S_{\mathsf{reject}}|]$
        If $j \leq p - \mathsf{Rng}(T)$ then
            $i \leftarrow i + 1$
            $z_i \leftarrow\!\!\$\ \mathbb{Z}_p \setminus \mathsf{Dom}(T)$          // $\mathcal{G}_{[1,3)}$
            $T[z_i] \leftarrow Z$                       // $\mathcal{G}_{[1,3)}$
            $\vec{z_i} \leftarrow\!\!\$\ \mathbb{Z}_p \setminus \mathsf{Dom}(T)[\vec{z}]$ // $\mathcal{G}_{[3,\infty)}$
            $T[\mathsf{Z}_i] \leftarrow Z$                       // $\mathcal{G}_{[3,\infty)}$
        Else $S_{\mathsf{reject}} \leftarrow S_{\mathsf{reject}} \cup \{Z\}$
If $\exists x, y\colon T[x] = X$ and $T[y] = Y$ then
    Return $\textsc{Exp}(ax + by)$
Return $\perp$
```

Fig. 30: Games $\mathcal{G}_{1-4}$ used in the proof of Proposition 7. We use S_{pre} to denote the set of strings in S that contain the prefix pre.

TRANSITION FROM $\mathcal{G}_1$ TO $\mathcal{G}_2$. Games $\mathcal{G}_1$ and $\mathcal{G}_2$ are identical until bad_0 is set, therefore

$$\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_1}].$$

BOUNDING $\Pr[\mathsf{bad}_0^{\mathcal{G}_1}]$. The bad_0 flag is set in $\mathcal{G}_1$ if the $\textsc{Exp}$ oracle samples a string containing the prefix pre from $S \setminus (\mathsf{Rng}(T) \cup S_{\mathsf{reject}})$. Game $\mathcal{G}_2$ adds an instruction that reassigns $T[x]$ to a value that does not contain the prefix pre. We use a counting argument to bound $\Pr[\mathsf{bad}_0^{\mathcal{G}_1}]$. Strings in S are k bits long. We use S_{pre} to denote the set of strings in S that contain the prefix pre. We have $|S_{\mathsf{pre}}| = 2^{k-|\mathsf{pre}|}$. Of these, $|(\mathsf{Rng}(T) \cup S_{\mathsf{reject}}) \cap S_{\mathsf{pre}}|$ have either already been added to $\mathsf{Rng}(T)$ or to S_{reject} during prior $\textsc{Op}$ queries. The probability that $\textsc{Exp}$ picked one of the remaining $2^{k-|\mathsf{pre}|} - |(\mathsf{Rng}(T) \cup |S_{\mathsf{reject}}|) \cap S_{\mathsf{pre}}|$ values is $(2^{k-|\mathsf{pre}|} - |(\mathsf{Rng}(T) \cup |S_{\mathsf{reject}}|) \cap S_{\mathsf{pre}}|)/(|S| - |\mathsf{Rng}(T)| - |S_{\mathsf{reject}}|)$. Hence, the probability that a query to Exp set bad_0 is bounded by

$$\frac{2^{k-|\mathsf{pre}|} - |(\mathsf{Rng}(T) \cup |S_{\mathsf{reject}}|) \cap S_{\mathsf{pre}}|}{|S| - |\mathsf{Rng}(T)| - |S_{\mathsf{reject}}|} \le \frac{2^{k-|\mathsf{pre}|}}{|S| - |\mathsf{Rng}(T)| - |S_{\mathsf{reject}}|}$$
$$\le \frac{2^{k-|\mathsf{pre}|} + |\mathsf{Rng}(T)| + |S_{\mathsf{reject}}|}{2^k} \le \frac{2^{k-|\mathsf{pre}|} + (q_{\text{Exp}} + q_{\text{Op}} + 2) + q_{\text{Op}}}{2^k}.$$

Applying a union bound over all Exp queries, we have

$$\Pr[\mathsf{bad}_0^{\mathcal{G}_1}] \le \frac{(q_{\text{Exp}} + q_{\text{Op}} + 2) \cdot (2^{k-|\mathsf{pre}|} + q_{\text{Exp}} + 2 \cdot q_{\text{Op}} + 2)}{2^k}.$$

Transition $\mathcal{G}_2$ to $\mathcal{G}_3$. In game $\mathcal{G}_2$ we update the table T such that it performs a "symbolic" simulation of the injection σ. Specifically, we introduce indeterminates Z_i so the table T is of the form $T \colon \mathbb{Z}_p[\mathsf{Z}_i] \to S$, i.e., the domain of T now consists of (linear) polynomials of the form $\mathcal{P}(\mathsf{Z}_i) = \alpha\mathsf{Z}_i + \beta$ for $\alpha, \beta \in \mathbb{Z}_p$. Note that the adversary can only make constant polynomial queries directly to Exp. However, it may indirectly query Exp on a non-constant polynomial via an Op query. Whenever a query Exp(x) is made in $\mathcal{G}_3$, the oracle checks if $\exists \mathcal{P} \in \mathsf{Dom}(T) \colon \mathcal{P}(\boldsymbol{z}) = x(\boldsymbol{z})$ where $\boldsymbol{z}$ is the vector of all $z_i \in \mathbb{Z}_p$ sampled so far. If the conditional evaluates to true, the oracle sets the bad_1 flag and sets $T[x]$ to $T[\mathcal{P}]$. This ensures that games $\mathcal{G}_2$ and $\mathcal{G}_3$ are identical. It follows that

$$\Pr[\mathcal{G}_2] = \Pr[\mathcal{G}_3].$$

Transition $\mathcal{G}_3$ to $\mathcal{G}_4$. Games $\mathcal{G}_3$ and $\mathcal{G}_4$ are identical until bad_1 is set, therefore

$$\Pr[\mathcal{G}_3] - \Pr[\mathcal{G}_4] \le \Pr[\mathsf{bad}_1^{\mathcal{G}_4}].$$

Game $\mathcal{G}_3$ removes the instruction that assigns $T[\mathcal{P}]$ to $T[x]$.

Bounding $\Pr[\mathsf{bad}_1^{\mathcal{G}_4}]$. We use the Schwartz-Zippel lemma to bound $\Pr[\mathsf{bad}_1^{\mathcal{G}_3}]$. Note that bad_1 is only set if $T[x] = \bot$ and $\exists \mathcal{P} \in \mathsf{Dom}(T) \colon \mathcal{P}(\boldsymbol{z}) = x(\boldsymbol{z})$. This implies that $x \ne \mathcal{P}$ and $x(\boldsymbol{z}) = \mathcal{P}(\boldsymbol{z})$ or $(\mathcal{P} - x)(\boldsymbol{z}) = 0$. As $\mathcal{P} - x$ is a linear polynomial and $\boldsymbol{z}$ is sampled uniformly at random from $\mathbb{Z}_p$, we use the Schwartz-Zippel lemma to bound the probability that a particular Exp query sets bad_1 as $\mathsf{Rng}(T)/p \le (q_{\text{Exp}} + q_{\text{Op}})/p$. Applying a union bound over all Exp queries, we have

$$\Pr[\mathsf{bad}_1^{\mathcal{G}_4}] \le (q_{\text{Exp}} + q_{\text{Op}} + 2)^2/p.$$

Bounding $\Pr[\mathcal{G}_4]$. The adversary cannot win $\mathcal{G}_4$. To win, $\mathcal{A}_{\mathsf{DL\text{-}PM}}$ must return a constant polynomial r and a bitstring R such that R begins with pre and $T[r] = R$ holds at the end of the execution of the game. This is impossible because values written into T by Exp never begin with pre and Op only writes to T for non-constant polynomials. Hence,

$$\Pr[\mathcal{G}_4] = 0.$$

C.2 Φ_{SP} Derives Messages from Hashed NE Keys

In this section, we justify our claim that the key cycle in SealPacket can be viewed as an instance of deriving the message from a hashed key. Let $\Phi_{\mathsf{SP}} = \mathsf{SIGENC\text{-}DER}[\mathsf{NE}, \mathsf{H}, \mathsf{DS}, m, n, sk] \cup \mathsf{ENC\text{-}DER}[m]$ be the message deriving function defined in Construction 5. Then we build functions γ and H for Φ_{SP} that satisfy Definition 4. The case of $\mathsf{ENC\text{-}DER}[m]$ is trivial: the corresponding γ is the constant function that always returns m (regardless of its input H). So we focus on the other case now. Let $\phi = \mathsf{SIGENC\text{-}DER}[\mathsf{NE}, \mathsf{H}, \mathsf{DS}, m, n, sk]$ for any $m \in \{0,1\}^*$, $n \in \{0,1\}^{\mathsf{NE.nl}}$, and $sk \in \{0,1\}^{\mathsf{DS.skl}}$. In Keybase, $\mathsf{NE} = \mathsf{XSalsa20\text{-}Poly1305}$, $\mathsf{H} = \mathsf{SHA\text{-}512}$, and $\mathsf{DS} = \mathsf{Ed25519}$. We provide the expanded definition of ϕ in Fig. 31 wherein $\mathbb{G}_p$ be the cyclic group underlying the Ed25519 scheme as specified in Construction 6. In particular, the order of $\mathbb{G}_p$ is p, the generator of $\mathbb{G}_p$ is $\mathbf{B}$, and the elements of $\mathbb{G}_p$ are encoded as 256-bit strings.

Note that the only lines within the message deriving function ϕ that use the input key K are the ones that are indicated with a $\boxed{\text{box}}$. Additionaly, we define r and ch to be the outputs of the SHA-512 calls modulo p. These values were likewise used only modulo p in the function ϕ; we merely made this explicit in Fig. 31.

We define the function H corresponding to ϕ in Fig. 31. It encapsulates the two $\mathsf{SHA\text{-}512} \bmod p$ instances within ϕ. In recent work [BDD23] formalized and showed the indifferentiability of $\mathsf{SHA\text{-}512}$ reduced modulo a sufficiently large prime p as a part of the broader analysis of the $\mathsf{Ed25519}$ scheme. If we show that the two $\mathsf{SHA\text{-}512} \bmod p$ instances in ϕ (namely H_0 and H_1) can be modeled as two *independent* random oracles, then we can directly apply the result of [BDD23] to prove that H can be reasonably treated as a random oracle. For this, it would suffice to show that (a) each $\mathsf{SHA\text{-}512}$ call takes inputs of a fixed length, and (b) the input lengths of these $\mathsf{SHA\text{-}512}$ calls are distinct. These properties follow from K, n, h, e_2, R, and A being fixed-length. With this definition of H, we can straightforwardly build γ such that $\phi = \gamma(\mathsf{H}(K))$, where H can be modeled as a random oracle.

```
$\phi[\mathsf{XSalsa20\text{-}Poly1305}, \mathsf{SHA\text{-}512}, \mathsf{Ed25519}, m, n, sk](K)$
$h \leftarrow \mathsf{SHA\text{-}512}(m)$ ; $\boxed{m_s \leftarrow \text{“Keybase-Chat-2”} \,\|\, \langle K, n, h \rangle}$
$e_1 \,\|\, e_2 \leftarrow \mathsf{SHA\text{-}512}(sk)$   // s.t. $|e_1| = |e_2| = 256$
$t \leftarrow 2^{k-2} + \sum_{i=3}^{k-3} 2^i \cdot e_1[i]$
$a \leftarrow t \bmod p$
$\boxed{r \leftarrow \mathsf{SHA\text{-}512}(e_2, m_s) \bmod p}$
$R \leftarrow r\mathbf{B}$ ; $A \leftarrow a\mathbf{B}$
$\boxed{\mathsf{ch} \leftarrow \mathsf{SHA\text{-}512}(R, A, m_s) \bmod p}$
$z \leftarrow (r + \mathsf{ch} \cdot a) \bmod p$
$s \leftarrow R \,\|\, z$
$m_e \leftarrow s \,\|\, m$ ; Return $m_e$
```

```
$\mathsf{H}(K, n, h, X)$
require $|n| = \mathsf{XSalsa20\text{-}Poly1305.nl}$
require $|h| = \mathsf{SHA\text{-}512.ol}$
require $|K| = \mathsf{XSalsa20\text{-}Poly1305.kl}$
require $|X| \in \{256, 512\}$
$m_s \leftarrow \text{“Keybase-Chat-2”} \,\|\, \langle K, n, h \rangle$
If $|X| = 256$ then
    Return $\mathsf{H}_0(X, m_s)$   // $\mathsf{H}_0(\cdot) = \mathsf{SHA\text{-}512}(\cdot) \bmod p$
Return $\mathsf{H}_1(X, m_s)$   // $\mathsf{H}_1(\cdot) = \mathsf{SHA\text{-}512}(\cdot) \bmod p$
```

Fig. 31: The function $\phi = \mathsf{SIGENC\text{-}DER}[\mathsf{NE}, \mathsf{H}, \mathsf{Ed25519}, m, n, sk]$. The code highlighted in gray was added by expanding the signing algorithm $\mathsf{Ed25519.Sig}$ (according to its definition in Fig. 25 of Appendix C.1).

<u>Challenges in Modelling H as a Random Oracle.</u> Here we enumerate the subtle issues with claiming that the function H defined above satisfies Definition 4.

First of all, according to Definition 4, the domain of the function H should be $\{0,1\}^{\mathsf{XSalsa20\text{-}Poly1305.kl}}$ where $\mathsf{XSalsa20\text{-}Poly1305.kl} = 256$. However, the functions H_0 and H_1 in Fig. 31 take inputs of the form (e_2, m_s) and (R, A, m_s) where $|e_2| = |R| = |A| = 256$ and $|m_s| = 1072$. This means that the domain D of H is $\{0,1\}^{256} \times (\{0,1\}^{1072} \cup \{0,1\}^{256+1072})$ and its range R is $\mathbb{Z}_p$. We fix this in the next subsection by defining a function $F : \{0,1\}^{\mathsf{XSalsa20\text{-}Poly1305}} \rightarrow \{f \,|\, f : D' \rightarrow R\}$ where $D' = \{0,1\}^{704} \times (\{0,1\}^{256} \cup \{0,1\}^{512})$ such that $F(K)(x) = H(K, x)$. Then we define γ_0 corresponding to F such that $\phi = \gamma_0(F(K))$. Since F is the composition of an injection and a function that can be modeled as a random oracle, F can also be modeled as a random oracle.

Secondly, we have thus far omitted discussing the two other $\mathsf{SHA\text{-}512}$ (not $\bmod p$) calls within ϕ. If we tried to formally reduce security to the indifferentiability result of [BDD23] these extra uses of $\mathsf{SHA\text{-}512}$ require small tweaks to the approach to handle. We provide an overview of these tweaks in the last subsection of this appendix.

<u>Per-Key Random Oracles.</u> Let $D' = \{0,1\}^{704} \times (\{0,1\}^{256} \cup \{0,1\}^{512})$ and $R = \mathbb{Z}_p$. Then we define the functions F and γ_0 as discussed above in Fig. 32. The function F takes an $\mathsf{XSalsa20\text{-}Poly1305}$ key $K \in \{0,1\}^{\mathsf{XSalsa20\text{-}Poly1305.kl}}$ as input, and it returns the description of another function $f[K] : D' \rightarrow R$. We emphasize that the output of $F(K)$ specifies the function $f[K]$ that can itself be evaluated on arbitrary input tuples (n, h, X). Note that there is domain separation. For $K \neq K'$, $f[K]$ and $f[K']$ will never call $\mathsf{SHA\text{-}512} \bmod p$ on the same inputs. Consequently, (modeling $\mathsf{SHA\text{-}512}$ mod p as a random oracle) each $f[K]$ is independent and random, so F can be viewed as a random oracle. We also define $\gamma_0[m, n, sk]$ corresponding to ϕ in Fig. 32. It is straightforward to see that $\phi = \mathsf{SIGENC\text{-}DER}[\mathsf{NE}, \mathsf{H}, \mathsf{DS}, m, n, sk]$ can be expressed as $\gamma_0(F(K))$. Except for the nuance we discuss in the next subsection, we have built the functions $\mathsf{H} = F$ and $\gamma = \gamma_0$ corresponding to the message deriving function ϕ such that they satisfy Definition 4.

```
$F(K)$   // $|K| = \ell$
Return $f[K]$
Function $f[K](n, h, X)$
    require $|n| = \mathsf{XSalsa20\text{-}Poly1305.nl}$ and $|h| = \mathsf{SHA\text{-}512.ol}$
    $m_s \leftarrow$ "Keybase-Chat-2" $\| \langle K, n, h \rangle$
    $o \leftarrow \mathsf{SHA\text{-}512}(X, m_s) \bmod p$
    Return $o$
```

```
$\gamma_0[m, n, sk](f)$
$h \leftarrow \mathsf{SHA\text{-}512}(m)$
$e_1 \| e_2 \leftarrow \mathsf{SHA\text{-}512}(sk)$
$t \leftarrow 2^{k-2} + \sum_{i=3}^{k-3} 2^i \cdot e_1[i]$
$a \leftarrow t \bmod p$
$r \leftarrow f(n, h, e_2)$
$R \leftarrow r\mathbf{B}$ ; $A \leftarrow a\mathbf{B}$
$\mathsf{ch} \leftarrow f(n, h, (R, A))$
$z \leftarrow (r + \mathsf{ch} \cdot a) \bmod p$
$s \leftarrow R \| z$
$m_e \leftarrow s \| m$ ; Return $m_e$
```

Fig. 32: **Left pane:** The function F corresponding to ϕ. **Right pane:** The function $\gamma_0[m, n, sk]$ corresponding to ϕ.

Proof Flow from Indifferentiability. As we mentioned earlier, the presence of the two plain (i.e. not reduced modulo p) SHA-512 calls in ϕ introduces subtleties that require us to tweak our steps so far. Recall that we want to show that ϕ derives messages from a hashed key and use this property to prove the KDMAE security of XSalsa20-Poly1305 with respect to ϕ. We now give an overview of the required tweaks and the overall proof flow that would be required to leverage the indifferentiability results of [BDD23]. In the following game transitions, we start with the game $\mathcal{G}_0$ that captures the KDMAE security of NE = XSalsa20-Poly1305 with respect to the message deriving function ϕ. We sketch intermediate game transitions to ultimately reach $\mathcal{G}_6$ that captures the KDMAE security of XSalsa20-Poly1305 with respect to the function $\gamma_1(F(.))$ where we define γ_1 along the way and F is a random oracle.

1. Game $\mathcal{G}_0$. In $\mathcal{G}_0$, $\mathcal{A}_{\mathsf{KDMAE}}$ internally calls the standard model function SHA-512 and picks ϕ = SIGENC-DER[XSalsa20-Poly1305, SHA-512, Ed25519, m, n, sk] which runs the standard model calls to both plain SHA-512 (of the form $h \leftarrow \mathsf{SHA\text{-}512}(m)$ and $e_1||e_2 \leftarrow \mathsf{SHA\text{-}512}(sk)$) and SHA-512 mod p.
2. Game $\mathcal{G}_1$. In $\mathcal{G}_1$, we replace the function ϕ with its modified version $\tilde{\phi}$. Unlike the former, the latter does not invoke plain SHA-512 but instead takes the values h and $e_1||e_2$ as part of its input. We then have $\mathcal{A}_{\mathsf{KDMAE}}$ compute the values h and $e_1||e_2$ locally by running SHA-512 before its calls to $\tilde{\phi}$. As this transition only involves reorganization of logic, $\mathcal{G}_1$ is equivalent to $\mathcal{G}_0$.
3. Game $\mathcal{G}_2$. In game $\mathcal{G}_2$, we note that SHA-512 is an MD-based hash function, computed by repeatedly invoking the compression function sha. We replace all calls to SHA-512 mod p with the MD construction mod p that using the compression function sha inside of adversary and the function ϕ. We replace all calls to plain SHA-512 with the MD construction using the compression function sha inside of adversary. So this step does not change the view of the adversary and hence $\mathcal{G}_2$ is equivalent to $\mathcal{G}_1$.
4. Game $\mathcal{G}_3$. Now we switch from the standard model to the ideal model and replace all instances of the compression function sha with the ideal function h. Here we assume that the compression function sha can be modeled as an idealized compression function. This heuristic step sets us for the indifferentiability analysis.
5. Game $\mathcal{G}_4$. In game $\mathcal{G}_4$, we use the indifferentiabilty results of [BDD23] to replace the MD mod p construction (with access to h) used by $\tilde{\phi}$ with the random oracle H. This changes all other queries to h (particularly, those internal to $\mathcal{A}_{\mathsf{KDMAE}}$ that arose from the calls to plain SHA-512) to instead use a stateful simulator with access to H to simulate h.
6. Game $\mathcal{G}_5$. In this game we embed the code for the simulator from the previous step inside $\mathcal{A}_{\mathsf{KDMAE}}$. So instead of having direct access to the simulator, the adversary runs it as part of its own code.[4] This step only constitutes syntactic changes and hence $\mathcal{G}_5$ is equivalent to $\mathcal{G}_4$.

[4] Incorporating the plain SHA-512 calls into the adversary $\mathcal{A}_{\mathsf{KDMAE}}$'s code in $\mathcal{G}_1$ was done as a set up for this step, where it is important that the simulator is only ever run internally to $\mathcal{A}_{\mathsf{KDMAE}}$ and not in the message derivation function. Otherwise we would have issues as the simulator is stateful.

7. Game $\mathcal{G}_6$ In our final transition, we replace H and $\tilde{\phi}$ with F and $\gamma_1(F(\cdot))$ respectively. Here F is as defined in Fig. 32 except $\mathsf{SHA\text{-}512}$ mod p is replaced with H and γ_1 is the same as γ_0 except it does not make calls to plain $\mathsf{SHA\text{-}512}$ and is parameterized by $[h, n, e_1||e_2]$. Thus we have shown that the message deriving function ϕ derives messages from a hashed key.

C.3 KDMAE for Messages Derived from a Hashed Key (Proof of Proposition 1)

Overview. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_4$ in Fig. 33. We establish the following claims.

- $\mathsf{Adv}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi}(\mathcal{A}_{\mathsf{KDMAE}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1$
- $\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$
- $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq q^2_{\textsc{NewHonGroup}}/2^{\mathsf{NE.kl}+1}$
- $\Pr[\mathcal{G}_1] = \Pr[\mathcal{G}_2]$
- $\Pr[\mathcal{G}_2] \leq \Pr[\mathcal{G}_3]$
- $\Pr[\mathcal{G}_3] \leq \Pr[\mathcal{G}_4] + \Pr[\mathsf{bad}_1^{\mathcal{G}_4}]$
- $\Pr[\mathsf{bad}_1^{\mathcal{G}_4}] \leq \mathsf{Adv}^{\mathsf{KR}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{KR}})$
- $2 \cdot \Pr[\mathcal{G}_4] - 1 \leq \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}})$

By combining the above, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi}(\mathcal{A}_{\mathsf{KDMAE}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1 &= 2 \cdot \left(\sum_{i=0}^{3} (\Pr[\mathcal{G}_i] - \Pr[\mathcal{G}_{i+1}]) + \Pr[\mathcal{G}_4] \right) - 1 \\ &\leq 2 \cdot (\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] + \Pr[\mathsf{bad}_1^{\mathcal{G}_4}] + \Pr[\mathcal{G}_4]) - 1 \\ &\leq q^2_{\textsc{NewHonGroup}}/2^{\mathsf{NE.kl}} + 2 \cdot \mathsf{Adv}^{\mathsf{KR}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{KR}}) + \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}}).\end{aligned}$$

We justify our claims in the following paragraphs.

Game $\mathcal{G}_0$. As per Proposition 1, consider game $\mathcal{G}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi}$ where Φ derives messages from a hashed key so every message-deriving function is of the form $\phi_\gamma(\mathsf{K}[g]) = \gamma(\mathsf{H}(\mathsf{K}[g]))$, where H is modeled as a random oracle H. We let $\mathcal{R}$ denote the random oracle's range. We rewrite this game in a functionally equivalent way, obtaining game $\mathcal{G}_0$ as our starting point in this proof. The highlighted code in $\mathcal{G}_0$ marks the changes that we introduced while rewriting $\mathcal{G}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi}$. The code highlighted in green is only used for transitions between games $\mathcal{G}_0$ through $\mathcal{G}_4$; we will explain this code later. The code highlighted in gray introduces the following changes: (1) it writes $\phi_b(\mathsf{K}[g])$ as $\gamma_b(\mathrm{H}(\mathsf{K}[g]))$, (2) it populates and updates the sets $S_{\mathsf{honest}}, S_{\mathsf{reused}}$ that contain "honest" and "reused" NE keys, and (3) it adds a simulated random oracle $\mathrm{H}^{\mathsf{key}}_{\mathsf{secret}}$ that is meant to be called only on inputs that are keys of honest groups.

We will now explain the meaning of these sets, and justify that they do not alter the functionality of the initial game. It will follow that

$$\mathsf{Adv}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi}(\mathcal{A}_{\mathsf{KDMAE}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1.$$

First, we introduce the set S_{honest} that contains all honest symmetric keys at any point in time. Whenever the adversary creates an honest group, the newly sampled key is added to S_{honest} and whenever it exposes a group key, the exposed key is removed from S_{honest}. If the $\textsc{NewHonGroup}$ oracle ever samples the same symmetric key for two different groups, we set the bad_0 flag. Next, we rewrite the random oracle H such that if it is queried on a (currently) honest key K, it delegates the lazy sampling of the response to the simulated oracle $\mathrm{H}^{\mathsf{key}}_{\mathsf{secret}}$. Otherwise, it responds by lazily sampling the output itself. Note that H and $\mathrm{H}^{\mathsf{key}}_{\mathsf{secret}}$ use different tables T_0 and T_1 for responding to queries. Whenever an honest key is exposed, on top of removing it from the set S_{honest}, the game updates the table T_0 to be consistent with the table T_1 by setting $T_0[\mathsf{K}[g]]$ to $\mathrm{H}^{\mathsf{key}}_{\mathsf{secret}}(\mathsf{K}[g])$. Note that a key K is exposed iff $T_0[K] \neq \bot$. If H is called on an exposed key K, it returns $T_0[K]$. If $\mathcal{A}_{\mathsf{KDMAE}}$ queries its $\textsc{NewCorrGroup}$ oracle on inputs (g, K) such that $\mathsf{K}[g] = \bot$ and $K \in S_{\mathsf{honest}}$, we call the key K a "reused" key. We maintain a set S_{reused} of reused keys and set the bad_1 flag whenever a reused key is queried to $\textsc{NewCorrGroup}$. We require that H answer a query on a reused key on its own rather than delegating it to the simulated random oracle $\mathrm{H}^{\mathsf{key}}_{\mathsf{secret}}$. We assume, without loss of generality, that the adversary makes all of its NewHonGroup queries before making any other queries. Lastly, we add conditional statements in the $\textsc{Enc}$ oracle that result in three branches. All three branches execute the same line of code that assigns to z the output of $\mathrm{H}(\mathsf{K}[g])$. None of these changes affect the functionality of $\mathcal{G}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi}$.

```
Games $\mathcal{G}_0$–$\mathcal{G}_4$
$b \leftarrow_{\$} \{0,1\}$ ; $b' \leftarrow_{\$} \mathcal{A}_{\mathsf{KDMAE}}^{\mathrm{G,Enc,Dec,H}}$
Return $b = b'$

$\mathrm{H}(K)$
If $K \in S_{\mathsf{honest}}$ and $K \notin S_{\mathsf{reused}}$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$   // $\mathcal{A}_{\mathsf{KR}}$ breaks KR of NE.
    Return $\mathrm{H}^{\mathsf{key}}_{\mathsf{secret}}(K)$   // $\mathcal{G}_{[0,2)}$
    Return $\mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}(\mathsf{group}[K])$   // $\mathcal{G}_{[2,4)}$
If $T_0[K] = \bot$ then $T_0[K] \leftarrow_{\$} \mathcal{R}$
Return $T_0[K]$

$\mathrm{H}^{\mathsf{key}}_{\mathsf{secret}}(K)$
If $T_1[K] = \bot$ then $T_1[K] \leftarrow_{\$} \mathcal{R}$
Return $T_1[K]$

$\mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}(g)$   // $\mathcal{G}_{[2,\infty)}$
If $T_1[g] = \bot$ then $T_1[g] \leftarrow_{\$} \mathcal{R}$   // $\mathcal{G}_{[2,\infty)}$
Return $T_1[g]$   // $\mathcal{G}_{[2,\infty)}$
```

```
$\mathrm{Enc}(g, n, \phi_0, \phi_1)$   // s.t. $\phi_b(\mathsf{K}[g]) = \gamma_b(\mathrm{H}(\mathsf{K}[g]))$
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 $\bot$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
If $\mathsf{K}[g] \in S_{\mathsf{reused}}$ then
    $z \leftarrow \mathrm{H}(\mathsf{K}[g])$   // Unreachable in $\mathcal{G}_4$.
Else
    If $\mathsf{K}[g] \in S_{\mathsf{honest}}$ then
        // Only reachable if $\neg\mathsf{group\_is\_corrupt}[g]$.
        $z \leftarrow \mathrm{H}(\mathsf{K}[g])$   // $\mathcal{G}_{[0,2)}$
        $z \leftarrow \mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}(g)$   // $\mathcal{G}_{[2,\infty)}$
    Else $z \leftarrow \mathrm{H}(\mathsf{K}[g])$   // Only reachable if $\mathsf{group\_is\_corrupt}[g]$.
$m_b \leftarrow \gamma_b(z)$ ; $c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_b)$
$N \leftarrow N \cup \{(g, n)\}$ ; $C \leftarrow C \cup \{(g, n, c)\}$ ; Return $c$

$\mathrm{Dec}(g, n, c)$
// This oracle is identical to the corresponding
// oracle in the KDMAE game for NE as per Fig. 18.
```

```
$\mathrm{NewHonGroup}(g)$
require $\mathsf{K}[g] = \bot$
$\mathsf{K}[g] \leftarrow_{\$} \{0,1\}^{\mathsf{NE.kl}}$
$S_{\mathsf{honest}} \leftarrow S_{\mathsf{honest}} \cup \{\mathsf{K}[g]\}$
If $\mathsf{group}[\mathsf{K}[g]] \neq \bot$ then
    // Birthday bound.
    $\mathsf{bad}_0 \leftarrow \mathsf{true}$
    $\mathsf{stop}(\mathsf{false})$   // $\mathcal{G}_{[1,3)}$
$\mathsf{group}[\mathsf{K}[g]] \leftarrow g$

$\mathrm{ExposeGroup}(g)$
require $\mathsf{K}[g] \neq \bot$ and $\neg\mathsf{chal}[g]$
$\mathsf{group\_is\_corrupt}[g] \leftarrow \mathsf{true}$
$S_{\mathsf{honest}} \leftarrow S_{\mathsf{honest}} \setminus \{\mathsf{K}[g]\}$
If $K \notin S_{\mathsf{reused}}$ then
    $T_0[\mathsf{K}[g]] \leftarrow \mathrm{H}^{\mathsf{key}}_{\mathsf{secret}}(\mathsf{K}[g])$   // $\mathcal{G}_{[0,2)}$
    $T_0[\mathsf{K}[g]] \leftarrow \mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}(g)$   // $\mathcal{G}_{[2,\infty)}$
Return $\mathsf{K}[g]$

$\mathrm{NewCorrGroup}(g, K)$
require $\mathsf{K}[g] = \bot$
$\mathsf{group\_is\_corrupt}[g] \leftarrow \mathsf{true}$
If $K \in S_{\mathsf{honest}}$ and $K \notin S_{\mathsf{reused}}$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$   // $\mathcal{A}_{\mathsf{KR}}$ breaks KR of NE.
    $S_{\mathsf{reused}} \leftarrow S_{\mathsf{reused}} \cup \{K\}$   // $\mathcal{G}_{[0,4)}$
    $T_0[K] \leftarrow \mathrm{H}^{\mathsf{key}}_{\mathsf{secret}}(K)$   // $\mathcal{G}_{[0,2)}$
    $T_0[K] \leftarrow \mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}(\mathsf{group}[K])$   // $\mathcal{G}_{[2,4)}$
$\mathsf{K}[g] \leftarrow K$
```

```
Adversary $\mathcal{A}_{\mathsf{KR}}^{\mathrm{G,Enc,Dec,Guess}}$
$b \leftarrow_{\$} \{0,1\}$
$b' \leftarrow_{\$} \mathcal{A}_{\mathsf{KDMAE}}^{\mathrm{SimG,SimEnc,SimDec,SimH}}$

$\mathrm{SimH}(K)$
$\mathrm{Guess}(K)$
If $T_0[K] = \bot$ then $T_0[K] \leftarrow_{\$} \mathcal{R}$
Return $T_0[K]$

$\mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}(g)$
If $T_1[g] = \bot$ then $T_1[g] \leftarrow_{\$} \mathcal{R}$
Return $T_1[g]$
```

```
$\mathrm{SimEnc}(g, n, \phi_0, \phi_1)$   // s.t. $\phi_b(\mathsf{K}[g]) = \gamma_b(\mathrm{H}(\mathsf{K}[g]))$
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 $\bot$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
If $\neg\mathsf{group\_is\_corrupt}[g]$ then $z \leftarrow \mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}(g)$ else $z \leftarrow \mathrm{SimH}(\mathsf{K}[g])$
$m_b \leftarrow \gamma_b(z)$ ; $c \leftarrow \mathrm{Enc}(g, n, m_b)$
$N \leftarrow N \cup \{(g, n)\}$ ; $C \leftarrow C \cup \{(g, n, c)\}$ ; Return $c$

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

```
$\mathrm{SimNewHonGroup}(g)$
require $\mathsf{K}[g] = \bot$
$\mathsf{K}[g] \leftarrow$ “dummy”
$\mathrm{NewHonGroup}(g)$

$\mathrm{SimExposeGroup}(g)$
require $\mathsf{K}[g] \neq \bot$ and $\neg\mathsf{chal}[g]$
$\mathsf{group\_is\_corrupt}[g] \leftarrow \mathsf{true}$
$\mathsf{K}[g] \leftarrow \mathrm{ExposeGroup}(g)$
$T_0[\mathsf{K}[g]] \leftarrow \mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}(g)$
Return $\mathsf{K}[g]$

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

Fig. 33: **Top pane:** Games $\mathcal{G}_0$–$\mathcal{G}_4$ for the proof of Proposition 1. The code highlighted in gray rewrites parts of game $\mathcal{G}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi}(\mathcal{A}_{\mathsf{KDMAE}})$ in an equivalent way. The code added for the transitions between games is highlighted in green. **Bottom pane:** Adversary $\mathcal{A}_{\mathsf{KR}}$ for the proof of Proposition 1. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_4$.

Transition from $\mathcal{G}_0$ to $\mathcal{G}_1$. In game $\mathcal{G}_1$, the NewHonGroup oracle executes the **stop**(false) instruction whenever bad_0 is set. This instruction immediately halts the game and returns false as its final output. As the games $\mathcal{G}_0$ and $\mathcal{G}_1$ are identical-until-bad, we have

$$\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}].$$

Bounding $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$. The probability that bad_0 is set in $\mathcal{G}_0$ is bounded by the probability that the NewHonGroup oracle assigns the same key K to two different groups. As keys are sampled uniformly at random from $\{0,1\}^{\mathsf{NE.kl}}$, we can use the birthday bound to get

$$\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq q^2_{\textsc{NewHonGroup}}/(2 \cdot 2^{\mathsf{NE.kl}}).$$

Transition from $\mathcal{G}_1$ to $\mathcal{G}_2$. We note that the change introduced in the previous game transition ensures that $\forall K \in S_{\mathsf{honest}} \setminus S_{\mathsf{reused}}$, there is a unique mapping between K and the corresponding group identifier g. Moreover, for every query $\mathrm{H}(K)$ for which $K \in S_{\mathsf{honest}} \setminus S_{\mathsf{reused}}$, the query is relayed to $\mathrm{H}^{\mathsf{key}}_{\mathsf{secret}}$. Now in $\mathcal{G}_2$ we use a new simulated oracle $\mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}$ to respond to all H queries satisfying $K \in S_{\mathsf{honest}} \setminus S_{\mathsf{reused}}$. In essence we switch from using $\mathrm{H}^{\mathsf{key}}_{\mathsf{secret}}$ to using $\mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}$. The unique mapping between group identifiers and keys enables us to make this switch without changing the functionality of $\mathcal{G}_1$. Hence,

$$\Pr[\mathcal{G}_1] = \Pr[\mathcal{G}_2].$$

Transition from $\mathcal{G}_2$ to $\mathcal{G}_3$. In game $\mathcal{G}_3$ the game no longer aborts when bad_0 is set. Note that this only increases the chances that the adversary wins the game. It follows that

$$\Pr[\mathcal{G}_2] \leq \Pr[\mathcal{G}_3].$$

Transition from $\mathcal{G}_3$ to $\mathcal{G}_4$. In game $\mathcal{G}_4$, the oracles H and NewCorrGroup do not execute any other instructions in the conditional branches that set the bad_1 flag. As games $\mathcal{G}_3$ and $\mathcal{G}_4$ are identical until bad_1 is set in $\mathcal{G}_4$, we have

$$\Pr[\mathcal{G}_3] - \Pr[\mathcal{G}_4] \leq \Pr[\mathsf{bad}_1^{\mathcal{G}_4}].$$

Bounding $\Pr[\mathsf{bad}_1^{\mathcal{G}_4}]$. To bound $\Pr[\mathsf{bad}_1^{\mathcal{G}_4}]$, we build an adversary $\mathcal{A}_{\mathsf{KR}}$ in Fig. 33 that simulates $\mathcal{G}_4$ for $\mathcal{A}_{\mathsf{KDMAE}}$. The adversary $\mathcal{A}_{\mathsf{KR}}$ uses its group oracles G to respond to group oracle queries made by $\mathcal{A}_{\mathsf{KDMAE}}$. The set S_{reused} is always empty in $\mathcal{G}_4$. So $\mathcal{A}_{\mathsf{KR}}$ can ignore the conditional that checks if $\mathsf{K}[g] \in S_{\mathsf{reused}}$ while simulating the Enc oracle. To simulate the other two conditional branches, $\mathcal{A}_{\mathsf{KR}}$ uses the $\mathsf{group\_is\_corrupt}$ table. Note that $\mathsf{K}[g] \in S_{\mathsf{honest}}$ iff $\neg\mathsf{group\_is\_corrupt}[g]$. It replaces calls to $\mathsf{NE.Enc}$ with its Enc oracle and calls to $\mathsf{NE.Dec}$ with its Dec oracle. Finally, $\mathcal{A}_{\mathsf{KR}}$ guesses every key queried to the random oracle H or the NewCorrGroup oracle. The bad_1 flag is only set in $\mathcal{G}_4$ if the adversary queried an honest key to either the random oracle H or to the NewCorrGroup oracle. By guessing such a key, $\mathcal{A}_{\mathsf{KR}}$ wins its game. Thus,

$$\Pr[\mathsf{bad}_1^{\mathcal{G}_4}] \leq \mathsf{Adv}^{\mathsf{KR}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{KR}}).$$

Bounding $\Pr[\mathcal{G}_4]$. We build an adversary $\mathcal{A}_{\mathsf{AEAD}}$ in Fig. 34 that perfectly simulates game $\mathcal{G}_4$ for $\mathcal{A}_{\mathsf{KDMAE}}$ and wins whenever $\mathcal{A}_{\mathsf{KDMAE}}$ does. It uses its group oracles G to simulate responses to group oracle queries made by $\mathcal{A}_{\mathsf{KDMAE}}$ and its oracles Enc and Dec to simulate the corresponding oracles in $\mathcal{G}_4$. Therefore,

$$2 \cdot \Pr[\mathcal{G}_4] - 1 \leq \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}}).$$

C.4 Out-Group AE Security of BoxMessage (Proof of Theorem 4)

Overview. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_3$ in Fig. 35. We establish the following claims.

- ○ $\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{BM},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{bwOAE\text{-}of\text{-}BM}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1$
- ○ $\Pr[\mathcal{G}_0] = \Pr[\mathcal{G}_1]$
- ○ $\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_1}]$
- ○ $\Pr[\mathsf{bad}_0^{\mathcal{G}_1}] \leq \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{CR}})$
- ○ $\Pr[\mathcal{G}_2] - \Pr[\mathcal{G}_3] \leq \Pr[\mathsf{bad}_1^{\mathcal{G}_2}]$
- ○ $\Pr[\mathsf{bad}_1^{\mathcal{G}_2}] = 0$
- ○ $2 \cdot \Pr[\mathcal{G}_3] - 1 \leq \mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SP},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}},\mathsf{func}^{\perp}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}})$

```
Adversary $\mathcal{A}_{\mathsf{AEAD}}^{\mathrm{G},\textsc{Enc},\textsc{Dec}}$
$b' \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{KDMAE}}^{\textsc{SimG},\textsc{SimEnc},\textsc{SimDec},\textsc{SimH}}$
Return $b'$

$\textsc{SimH}(K)$
If $T_0[K] = \bot$ then $T_0[K] \leftarrow\!\!\$\ \mathcal{R}$
Return $T_0[K]$

$\mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}(g)$
If $T_1[g] = \bot$ then $T_1[g] \leftarrow\!\!\$\ \mathcal{R}$
Return $T_1[g]$
```

```
$\textsc{SimEnc}(g, n, \phi_0, \phi_1)$  // s.t. $\phi_b(\mathsf{K}[g]) = \gamma_b(\mathrm{H}(\mathsf{K}[g]))$
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 $\bot$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
If $\neg\mathsf{group\_is\_corrupt}[g]$ then $z \leftarrow \mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}(g)$ else $z \leftarrow \textsc{SimH}(\mathsf{K}[g])$
If $\phi_0 \neq \phi_1$ then
    $m_0 \leftarrow \gamma_0(z)$ ; $m_1 \leftarrow \gamma_1(z)$ ; $c \leftarrow \textsc{Enc}(g, n, m_0, m_1)$
Else
    $m \leftarrow \gamma_0(z)$ ; $c \leftarrow \textsc{Enc}(g, n, m, m)$
$N \leftarrow N \cup \{(g, n)\}$ ; $C \leftarrow C \cup \{(g, n, c)\}$ ; Return $c$

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

```
$\textsc{SimNewHonGroup}(g)$
require $\mathsf{K}[g] = \bot$
$\mathsf{K}[g] \leftarrow$ "dummy"
$\textsc{NewHonGroup}(g)$
```

```
$\textsc{SimExposeGroup}(g)$
require $\mathsf{K}[g] \neq \bot$ and $\neg\mathsf{chal}[g]$
$\mathsf{group\_is\_corrupt}[g] \leftarrow \mathsf{true}$
$\mathsf{K}[g] \leftarrow \textsc{ExposeGroup}(g)$
$T_0[\mathsf{K}[g]] \leftarrow \mathrm{H}^{\mathsf{group}}_{\mathsf{secret}}(g)$
Return $\mathsf{K}[g]$
```

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

Fig. 34: Adversary $\mathcal{A}_{\mathsf{AEAD}}$ for the proof of Proposition 1. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_4$.

```
Games $\mathcal{G}_0$–$\mathcal{G}_3$
$b \leftarrow\!\!\$\, \{0,1\}$ ; $b' \leftarrow\!\!\$\, \mathcal{A}_{\mathsf{bwOAE\text{-}of\text{-}BM}}^{\text{U,G,SigEnc,VerDec}}$ ; Return $b = b'$

SigEnc$(g, u, n, m_{\mathsf{body},0}, m_{\mathsf{body},1}, ad)$
require $\mathsf{K}[g] \neq \bot$ and $|m_{\mathsf{body},0}| = |m_{\mathsf{body},1}|$
require $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$(n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n$ ; If $n_{\mathsf{body}} = n_{\mathsf{header}}$ then return $\bot$
If $(g, n_{\mathsf{body}}) \in N$ or $(g, n_{\mathsf{header}}) \in N$ then return $\bot$
If $m_{\mathsf{body},0} \neq m_{\mathsf{body},1}$ then
    If $\mathsf{group\_is\_corrupt}[g]$ then return $\bot$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$c_{\mathsf{body}} \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n_{\mathsf{body}}, m_{\mathsf{body},b})$
$h_{\mathsf{body}} \leftarrow \mathsf{H}(n_{\mathsf{body}} \| c_{\mathsf{body}})$ ; $m_{\mathsf{header}} \leftarrow \langle ad, u, g, h_{\mathsf{body}} \rangle$
$c_{\mathsf{header}} \leftarrow \mathsf{SP.SigEnc}(g, \mathsf{K}[g], u, \mathsf{sk}[u], n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon)$
$C_{\mathsf{SP}} \leftarrow C_{\mathsf{SP}} \cup \{((g, u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}})\}$
$c \leftarrow (c_{\mathsf{body}}, c_{\mathsf{header}})$ ; $N \leftarrow N \cup \{(g, n_{\mathsf{header}}), (g, n_{\mathsf{body}})\}$
$C \leftarrow C \cup \{((g, u, n, m_{\mathsf{body},b}, ad), c)\}$
$W[g, u, n, c, ad] \leftarrow m_{\mathsf{body},1}$ ; $T[g, n_{\mathsf{header}}] \leftarrow (n_{\mathsf{body}}, c_{\mathsf{body}})$
$Q \leftarrow Q \cup \{((g, u, n, m_{\mathsf{body},0}, m_{\mathsf{body},1}, ad), c)\}$ ; Return $c$

NewHonUser$(u)$
NewHonGroup$(g, \mathsf{users})$
ExposeUser$(u)$
ExposeGroup$(g)$
NewCorrUser$(u, sk, vk)$
NewCorrGroup$(g, K, \mathsf{users})$
// These oracles are identical to the corresponding
// oracles in the OAE game for BM as per Fig. 7.
```

```
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 $u \in \mathsf{members}[g]$
If $W[g, u, n, c, ad] \neq \bot$ then return $\bot$                    // $\mathcal{G}_{[1,\infty)}$
$(n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n$ ; $(c_{\mathsf{body}}, c_{\mathsf{header}}) \leftarrow c$
$m_{\mathsf{header}} \leftarrow \mathsf{SP.VerDec}(g, \mathsf{K}[g], u, \mathsf{vk}[u], n_{\mathsf{header}}, c_{\mathsf{header}}, \varepsilon)$
$h_{\mathsf{body}} \leftarrow \mathsf{H}(n_{\mathsf{body}} \| c_{\mathsf{body}})$
If $m_{\mathsf{header}} \neq \langle ad, u, g, h_{\mathsf{body}} \rangle$ then return $\bot$
$m_{\mathsf{body}} \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n_{\mathsf{body}}, c_{\mathsf{body}})$
If $m_{\mathsf{body}} = \bot$ then return $\bot$
$z \leftarrow ((g, u, n, m_{\mathsf{body}}, ad), c)$                    // $\mathcal{G}_{[0,1)}$
If $z \in C$ then return $\bot$                    // $\mathcal{G}_{[0,1)}$
If $b = 0$ then return $\bot$
If $\exists u'$: $((g, u', n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}}) \in C_{\mathsf{SP}}$ then
    If $(n_{\mathsf{body}}, c_{\mathsf{body}}) \neq T[g, n_{\mathsf{header}}]$ then
        $\mathsf{bad}_0 \leftarrow \mathsf{true}$   // $\mathcal{A}_{\mathsf{CR}}$ breaks CR of H.
        Return $m_{\mathsf{body}}$                    // $\mathcal{G}_{[0,2)}$
        Return $\bot$                    // $\mathcal{G}_{[2,\infty)}$
    Else   // $(n_{\mathsf{body}}, c_{\mathsf{body}}) = T[g, n_{\mathsf{header}}]$
        $\mathsf{bad}_1 \leftarrow \mathsf{true}$   // Unreachable in $\mathcal{G}_2$.
        Return $m_{\mathsf{body}}$                    // $\mathcal{G}_{[0,3)}$
        Return $\bot$                    // $\mathcal{G}_{[3,\infty)}$
Else   // $\nexists u'$: $((g, u', n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}}) \in C_{\mathsf{SP}}$
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$   // $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ breaks OAE[Enc] of SP.
    Return $m_{\mathsf{body}}$
```

```
Adversary $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}^{\text{U,G,SigEnc,VerDec,Enc}}$
$b' \leftarrow\!\!\$\, \mathcal{A}_{\mathsf{bwOAE\text{-}of\text{-}BM}}^{\text{SimU,SimG,SimSigEnc,SimVerDec}}$ ; Return $b'$

SimSigEnc$(g, u, n, m_{\mathsf{body},0}, m_{\mathsf{body},1}, ad)$
require $\mathsf{K}[g] \neq \bot$ and $|m_{\mathsf{body},0}| = |m_{\mathsf{body},1}|$
require $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$(n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n$ ; If $n_{\mathsf{body}} = n_{\mathsf{header}}$ then return $\bot$
If $(g, n_{\mathsf{body}}) \in N$ or $(g, n_{\mathsf{header}}) \in N$ then return $\bot$
If $m_{\mathsf{body},0} \neq m_{\mathsf{body},1}$ then
    If $\mathsf{group\_is\_corrupt}[g]$ then return $\bot$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$c_{\mathsf{body}} \leftarrow$ Enc$(g, n_{\mathsf{body}}, m_{\mathsf{body},0}, m_{\mathsf{body},1})$
$h_{\mathsf{body}} \leftarrow \mathsf{H}(n_{\mathsf{body}} \| c_{\mathsf{body}})$ ; $m_{\mathsf{header}} \leftarrow \langle ad, u, g, h_{\mathsf{body}} \rangle$
$c_{\mathsf{header}} \leftarrow$ SigEnc$(g, u, n_{\mathsf{header}}, m_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon)$
$c \leftarrow (c_{\mathsf{body}}, c_{\mathsf{header}})$ ; $N \leftarrow N \cup \{(g, n_{\mathsf{header}}), (g, n_{\mathsf{body}})\}$
$W[g, u, n, c, ad] \leftarrow m_{\mathsf{body},1}$ ; Return $c$
```

```
SimVerDec$(g, u, n, c, ad)$
require $\mathsf{K}[g] \neq \bot$ and $\neg\mathsf{group\_is\_corrupt}[g]$
require $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
If $W[g, u, n, c, ad] \neq \bot$ then return $\bot$
$(n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n$ ; $(c_{\mathsf{body}}, c_{\mathsf{header}}) \leftarrow c$
$m_{\mathsf{header}} \leftarrow$ VerDec$(g, u, n_{\mathsf{header}}, c_{\mathsf{header}}, \varepsilon)$
If $m_{\mathsf{header}} \neq \bot$ then **abort**$(1)$
Return $\bot$
```

```
SimNewHonGroup$(g, \mathsf{users})$
require $\mathsf{K}[g] = \bot$
$\mathsf{K}[g] \leftarrow$ "dummy"
NewHonGroup$(g)$
$\mathsf{members}[g] \leftarrow \mathsf{users}$

SimNewHonUser$(u)$
require $\mathsf{sk}[u] = \mathsf{vk}[u] = \bot$
$\mathsf{vk}[u] \leftarrow$ NewHonUser$(u)$
$\mathsf{sk}[u] \leftarrow$ ExposeUser$(u)$ ; Return $\mathsf{vk}[u]$
```

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

SimExposeUser$(u)$
require $\mathsf{sk}[u] \neq \bot$
Return $\mathsf{sk}[u]$
```

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

SimNewCorrUser$(u, sk, vk)$
require $\mathsf{sk}[u] = \mathsf{vk}[u] = \bot$
NewCorrUser$(u, sk, vk)$
$\mathsf{sk}[u] \leftarrow sk$ ; $\mathsf{vk}[u] \leftarrow vk$
```

Fig. 35: **Top pane:** Games $\mathcal{G}_0$–$\mathcal{G}_3$ for the proof of Theorem 4. The code highlighted in gray was added by expanding the symmetric signcryption algorithms $\mathsf{BM.SigEnc}, \mathsf{BM.VerDec}$, and the ciphertext-triviality predicate $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$ in game $\mathcal{G}_{\mathsf{BM},\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}},\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}}^{\mathsf{OAE}}(\mathcal{A}_{\mathsf{bwOAE\text{-}of\text{-}BM}})$. The code added for the transitions between games is highlighted in green. **Bottom pane:** Adversary $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ for the proof of Theorem 4. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_3$.

By combining the above, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{BM},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{bwOAE\text{-}of\text{-}BM}}) &= 2\cdot\Pr[\mathcal{G}_0]-1 = 2\cdot\Pr[\mathcal{G}_1]-1\\ &= 2\cdot\left(\sum_{i=0}^{1}(\Pr[\mathcal{G}_{i+1}]-\Pr[\mathcal{G}_{i+2}])+\Pr[\mathcal{G}_3]\right)-1\\ &= 2\cdot\left(\sum_{i=0}^{1}(\Pr[\mathsf{bad}_i^{\mathcal{G}_{i+1}}])+\Pr[\mathcal{G}_3]\right)-1\\ &\leq 2\cdot\mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{H}})+\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SP},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}},\mathsf{func}^{\perp}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}).\end{aligned}$$

We justify our claims in the following paragraphs.

Game $\mathcal{G}_0$. The game $\mathcal{G}_0$ is functionally equivalent to $\mathcal{G}^{\mathsf{OAE}}_{\mathsf{BM},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}$. Additionally, the former includes code lines highlighted in green that we use for transitions between games. Note that the `if-then-else` statement that was added in oracle VerDec returns m_{body} under all conditional branches. These changes do not affect the functionality of $\mathcal{G}_0$ and we have

$$\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{BM},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{bwOAE\text{-}of\text{-}BM}}) = 2\cdot\Pr[\mathcal{G}_0]-1.$$

Transition from $\mathcal{G}_0$ to $\mathcal{G}_1$. In the game $\mathcal{G}_0$ if the adversary forwards an honestly generated ciphertext from SigEnc to VerDec, then the condition $z \in C$ holds and VerDec will always return $\perp$. In game $\mathcal{G}_1$ we remove the conditional that checks if $z \in C$ from $\mathcal{G}_1$ and use the table W instead to achieve the same behavior without actually having to decrypt forwarded ciphertexts. For every SigEnc query, the table W indexed by the tuple (g, u, n, c, ad) stores the challenge message $m_{\mathsf{body},1}$ where c is the ciphertext returned for that query. This table W is then used to respond to VerDec queries on trivially forwarded ciphertexts. Note that the decryption correctness of NE and SP gives $W[g, u, n, c, ad] \neq \perp \iff ((g, u, n, m_{\mathsf{body}}, ad), c) \in C$ i.e. $W[g, u, n, c, ad] \neq \perp \iff z \in C$ holds in $\mathcal{G}_0$. However, if the condition $W[g, u, n, c, ad] \neq \perp$ evaluates to true in $\mathcal{G}_0$, the VerDec oracle immediately returns $\perp$. This means that the conditional checking whether $z \in C$ is unreachable in $\mathcal{G}_0$, and removing it in $\mathcal{G}_1$ does not change the output of VerDec. It follows that

$$\Pr[\mathcal{G}_0] = \Pr[\mathcal{G}_1].$$

Transitions from game $\mathcal{G}_1$ to $\mathcal{G}_3$. Let $i \in \{0, 1\}$. Games $\mathcal{G}_{i+1}$ and $\mathcal{G}_{i+2}$ are identical until bad_i is set. Each of these transitions removes an instruction returning m_{body} and returns $\perp$ instead. Therefore

$$\Pr[\mathcal{G}_{i+1}] - \Pr[\mathcal{G}_{i+2}] \leq \Pr[\mathsf{bad}_i^{\mathcal{G}_{i+1}}].$$

Bounding $\Pr[\mathsf{bad}_0^{\mathcal{G}_1}]$. We describe adversary $\mathcal{A}_{\mathsf{CR}}$ against the collision resistance of H that simulates game $\mathcal{G}_1$ for $\mathcal{A}_{\mathsf{bwOAE\text{-}of\text{-}BM}}$. Adversary $\mathcal{A}_{\mathsf{CR}}$ runs the code of $\mathcal{G}_1$ by sampling and maintaining all user and group keys locally. Consider a VerDec query that sets the bad_0 flag. During this query VerDec computed $h_{\mathsf{body}} = \mathsf{H}(n_{\mathsf{body}} \,\|\, c_{\mathsf{body}})$. Now, the bad_1 flag is only set when $\exists u' \colon ((g, u', n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}}) \in C_{\mathsf{SP}}$ and $(n_{\mathsf{body}}, c_{\mathsf{body}}) \neq T[g, n_{\mathsf{header}}]$. The former implies that there was a prior call to SigEnc during which $\mathsf{SP.SigEnc}$ was invoked on inputs $(g, u', n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon)$ and produced output c_{header}. Let the body ciphertext for this query be c'_{body}. We know that for the current VerDec query $z \notin C$. Therefore $c_{\mathsf{body}} \neq c'_{\mathsf{body}}$. Note that the message m_{header} uniquely encodes $h_{\mathsf{body}} = \mathsf{H}(n_{\mathsf{body}} \,\|\, c'_{\mathsf{body}})$. By the uniqueness of (g, n_{header}) we know that $T[g, n_{\mathsf{header}}]$ uniquely maps to the pair $(n_{\mathsf{body}} \,\|\, c'_{\mathsf{body}})$. Then we get $h_{\mathsf{body}} = \mathsf{H}(n_{\mathsf{body}} \,\|\, c_{\mathsf{body}}) = \mathsf{H}(n_{\mathsf{body}} \,\|\, c_{\mathsf{body}})'$ and $(n_{\mathsf{body}}, c_{\mathsf{body}}) \neq (n_{\mathsf{body}}, c'_{\mathsf{body}})$. Therefore, by returning $(n_{\mathsf{body}} \,\|\, c_{\mathsf{body}}, n_{\mathsf{body}} \,\|\, c'_{\mathsf{body}})$, adversary $\mathcal{A}_{\mathsf{CR}}$ wins the collision-resistance game of H whenever bad_0 is set. This gives

$$\Pr[\mathsf{bad}_0^{\mathcal{G}_1}] \leq \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}_{\mathsf{CR}}).$$

Bounding $\Pr[\mathsf{bad}_1^{\mathcal{G}_2}]$. The conditions that set bad_1 in $\mathcal{G}_2$ are $\exists u' \colon ((g, u', n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}}) \in C_{\mathsf{SP}}$ and $(n_{\mathsf{body}}, c_{\mathsf{body}}) = T[g, n_{\mathsf{header}}]$. Since m_{header} encodes u, we get $u' = u$. Then we can infer that $((g, u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}}) \in C_{\mathsf{SP}}$. This means that there was a prior call to SigEnc during which

$\mathsf{SP.SigEnc}$ was invoked on inputs $(g, u, n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon)$ and produced output c_{header}. By the construction of m, we know that the values ad and h_{body} were the same across a prior SigEnc query and the current VerDec query. The condition $(n_{\mathsf{body}}, c_{\mathsf{body}}) = T[g, n_{\mathsf{header}}]$ implies that $(n_{\mathsf{body}}, c_{\mathsf{body}})$ were also repeated across the two queries. Combining the above, we get that the values g, u, n, c, and ad were repeated, and hence $W[g, u, n, c, ad] \neq \bot$. Therefore it is impossible to set bad_1 in $\mathcal{G}_2$ and

$$\Pr[\mathsf{bad}_1^{\mathcal{G}_2}] = 0$$

Bounding $\Pr[\mathcal{G}_3]$. In Fig. 35 we show adversary $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ against the OAE security of SP that simulates the game $\mathcal{G}_3$ for adversary $\mathcal{A}_{\mathsf{OAE}}$ until bad_2 is set and wins whenever $\mathcal{A}_{\mathsf{OAE}}$ does. It simulates the user oracles using the corresponding oracles from its OAE security game and simulates responses to the group oracles using its group oracles. While simulating the SigEnc oracle for $\mathcal{A}_{\mathsf{OAE}}$, it replaces calls to $\mathsf{SP.SigEnc}$ with its SigEnc oracle and analogously, while simulating VerDec, it substitutes calls to $\mathsf{SP.VerDec}$ with its VerDec oracle. To simulate calls to $\mathsf{NE.Enc}$, adversary $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ uses its Enc oracle. Finally, if the bad_2 flag is never set and $\mathcal{A}_{\mathsf{OAE}}$ returns b', adversary $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ also returns b'. Now consider the case that bad_2 is set in the simulated game. It must be the case that $\nexists u' \colon ((g, u', n_{\mathsf{header}}, m_{\mathsf{header}}, \varepsilon), c_{\mathsf{header}}) \in C_{\mathsf{SP}}$ and $m_{\mathsf{header}} \neq \bot$. This is only possible when $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ has successfully forged a ciphertext and the bit b in its game is 1. Therefore $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ executes $\mathbf{abort}(1)$ and wins its game whenever bad_2 is set, giving

$$2 \cdot \Pr[\mathcal{G}_1] - 1 \leq \mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SP}, \mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}, \mathsf{func}^{\bot}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}).$$

C.5 Out-group AE Security of SealPacket (Proof of Theorem 5)

Overview. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_3$ in Fig. 36. We establish the following claims.

- $\mathsf{Adv}^{\mathsf{OAE}[\text{Enc}]}_{\mathsf{SP}, \mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}, \mathsf{func}^{\bot}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1$
- $\Pr[\mathcal{G}_0] = \Pr[\mathcal{G}_1]$
- $\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_1}]$
- $\Pr[\mathsf{bad}_0^{\mathcal{G}_1}] \leq \mathsf{Adv}^{\mathsf{SPARSE}}_{\mathsf{DS}, \mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}})$
- $\Pr[\mathcal{G}_2] = \Pr[\mathcal{G}_3]$
- $2 \cdot \Pr[\mathcal{G}_3] - 1 \leq \mathsf{Adv}^{\mathsf{KDMAE}}_{\mathsf{NE}, \Phi_{\mathsf{SP}}}(\mathcal{A}_{\mathsf{AEAD}})$

By combining the above, we have

$$\begin{aligned} \mathsf{Adv}^{\mathsf{wOAE}[\text{Enc}[\mathcal{M}, \mathsf{NE}]]}_{\mathsf{SP}, \mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}, \mathsf{func}^{\bot}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}) &= 2 \cdot ((\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2]) + \Pr[\mathcal{G}_2]) - 1 \\ &\leq 2 \cdot (\Pr[\mathsf{bad}_0^{\mathcal{G}_1}] + \Pr[\mathcal{G}_3]) - 1 \\ &\leq 2 \cdot \mathsf{Adv}^{\mathsf{SPARSE}}_{\mathsf{DS}, \mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}}) + \mathsf{Adv}^{\mathsf{KDMAE}}_{\mathsf{NE}, \Phi_{\mathsf{SP}}}(\mathcal{A}_{\mathsf{AEAD}}). \end{aligned}$$

We justify our claims in the following paragraphs.

Game $\mathcal{G}_0$. We claim that the view of adversary $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ in game $\mathcal{G}_0$ is identical to its view in game $\mathcal{G}^{\mathsf{OAE}[\text{Enc}]}_{\mathsf{SP}, \mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}, \mathsf{func}^{\bot}_{\mathsf{out}}}$. We use green highlighting to show code that was added for transitions between games. In particular, we add the set C_{NE} in SigEnc and the set L in Enc. The set C_{NE} stores the tuple (g, n, c) for every SigEnc query where c is the output of $\mathsf{NE.Enc}$ for that query. The set L stores the tuple $(g, n_{\mathsf{body}}, c_{\mathsf{body}})$ for every valid Enc query where c_{body} is the output of Enc for that query. The latter set is then used in VerDec to check whether the adversary forwarded a ciphertext returned by Enc. If that is the case then the bad_0 flag is set. This does not affect the functionality of the game. Therefore,

$$\mathsf{Adv}^{\mathsf{OAE}[\text{Enc}]}_{\mathsf{SP}, \mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}, \mathsf{func}^{\bot}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1.$$

Transition from $\mathcal{G}_0$ to $\mathcal{G}_1$. In game $\mathcal{G}_1$, we use the set C_{NE} in VerDec to check if $(g, n, c) \in C_{\mathsf{NE}}$ for a query (g, u, n, c, ad). If the condition is true, the oracle returns $\bot$. Now, $(g, n, c) \in C_{\mathsf{NE}} \iff \exists u' \colon ((g, u', n, m, ad), c) \in C$ where m is the unsigncryption of c. This implies that $(g, n, c) \in C_{\mathsf{NE}} \iff \mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}(z, C) = \mathsf{true}$. By definition of the output-guarding function $\mathsf{func}^{\bot}_{\mathsf{out}}$, the game $\mathcal{G}_0$ returns $\bot$ whenever $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}(z, C) = \mathsf{true}$. The game $\mathcal{G}_1$ returns $\bot$ whenever $(g, n, c) \in C_{\mathsf{NE}}$. Therefore the outputs of games $\mathcal{G}_0$ and $\mathcal{G}_1$ are consistent and we have

$$\Pr[\mathcal{G}_0] = \Pr[\mathcal{G}_1].$$

```
Games $\mathcal{G}_0$–$\mathcal{G}_3$
$b \leftarrow\!\!\$ \{0,1\}$ ; $b' \leftarrow\!\!\$ \mathcal{A}_{\text{wOAE-of-SP}}^{\text{U,G,SigEnc,VerDec,Enc}}$ ; Return $b = b'$

SigEnc($g, u, n, m_0, m_1, ad$)  // $ad = \varepsilon$
require $\mathsf{K}[g] \neq \perp$ and $|m_0| = |m_1|$
require $\mathsf{sk}[u] \neq \perp$ and $u \in \mathsf{members}[g]$ and $(g, n) \notin N$
If $m_0 \neq m_1$ then
    If $\mathsf{group\_is\_corrupt}[g]$ then return $\perp$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$h \leftarrow \mathsf{H}(m_b)$ ; $m_s \leftarrow$ "Keybase-Chat-2" $\|\ \langle \mathsf{K}[g], n, h \rangle$
$s \leftarrow \mathsf{DS.Sig}(\mathsf{sk}[u], m_s)$ ; $m_e \leftarrow s \,\|\, m_b$
$c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_e)$
$C_{\mathsf{NE}} \leftarrow C_{\mathsf{NE}} \cup \{(g, n, c)\}$ ; $N \leftarrow N \cup \{(g, n)\}$
$C \leftarrow C \cup \{((g, u, n, m_b, ad), c)\}$ ; Return $c$

Enc($g, n_{\mathsf{body}}, m_{\mathsf{body},0}, m_{\mathsf{body},1}$)
require $\mathsf{K}[g] \neq \perp$ 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 $\perp$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$c_{\mathsf{body}} \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n_{\mathsf{body}}, m_{\mathsf{body},b})$
$L \leftarrow L \cup \{(g, n_{\mathsf{body}}, c_{\mathsf{body}})\}$
$N \leftarrow N \cup \{(g, n_{\mathsf{body}})\}$ ; Return $c_{\mathsf{body}}$
```

```
VerDec($g, u, n, c, ad$)  // $ad = \varepsilon$
require $\mathsf{K}[g] \neq \perp$ and $\neg\mathsf{group\_is\_corrupt}[g]$
require $\mathsf{vk}[u] \neq \perp$ and $u \in \mathsf{members}[g]$
If $(g, n, c) \in C_{\mathsf{NE}}$ then return $\perp$                // $\mathcal{G}_{[1,\infty)}$
If $(g, n, c) \in L$ then return $\perp$                          // $\mathcal{G}_{[3,\infty)}$
$m_e \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c)$ ; 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" $\|\ \langle \mathsf{K}[g], n, h \rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], m_s, s)$ then return $\perp$
$z \leftarrow ((g, u, n, m, ad), c)$
If $\exists u' : ((g, u', n, m, ad), c) \in C$ then return $\perp$
If $(g, n, c) \in L$ then
    $\mathsf{bad}_0 \leftarrow \mathsf{true}$  // $\mathcal{A}_{\mathsf{SPARSE}}$ breaks SPARSE of DS.
    Return $\perp$                                           // $\mathcal{G}_{[2,\infty)}$
If $b = 0$ then return $\perp$ else return $m$

NewHonUser($u$)
NewHonGroup($g$, users)
ExposeUser($u$)
ExposeGroup($g$)
NewCorrUser($u, sk, vk$)
NewCorrGroup($g, K$, users)
// These oracles are identical to the corresponding
// oracles in the OAE game for SP as per Fig. 7.
```

```
Adversary $\mathcal{A}_{\mathsf{KDMAE}}^{\text{G,Enc,Dec}}$
$b' \leftarrow\!\!\$ \mathcal{A}_{\text{wOAE-of-SP}}^{\text{SimU,SimG,SimSigEnc,SimVerDec,SimEnc}}$ ; Return $b'$

SimSigEnc($g, u, n, m_0, m_1, ad$)  // $ad = \varepsilon$
require $\mathsf{K}[g] \neq \perp$ and $|m_0| = |m_1|$
require $\mathsf{sk}[u] \neq \perp$ and $u \in \mathsf{members}[g]$ and $(g, n) \notin N$
If $m_0 \neq m_1$ then
    If $\mathsf{group\_is\_corrupt}[g]$ then return $\perp$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$f_0 \leftarrow \mathsf{SIGENC\text{-}DER}[\mathsf{NE}, \mathsf{H}, \mathsf{DS}, m_0, n, \mathsf{sk}[u]]$
$f_1 \leftarrow \mathsf{SIGENC\text{-}DER}[\mathsf{NE}, \mathsf{H}, \mathsf{DS}, m_1, n, \mathsf{sk}[u]]$
$c \leftarrow \text{Enc}(g, n, f_0, f_1)$ ; $N \leftarrow N \cup \{(g, n)\}$ ; Return $c$

SimEnc($g, n_{\mathsf{body}}, m_{\mathsf{body},0}, m_{\mathsf{body},1}$)
require $\mathsf{K}[g] \neq \perp$ 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 $\perp$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$f_0 \leftarrow \mathsf{ENC\text{-}DER}[m_{\mathsf{body},0}]$
$f_1 \leftarrow \mathsf{ENC\text{-}DER}[m_{\mathsf{body},1}]$ ; $c_{\mathsf{body}} \leftarrow \text{Enc}(g, n_{\mathsf{body}}, f_0, f_1)$
$N \leftarrow N \cup \{(g, n_{\mathsf{body}})\}$ ; Return $c_{\mathsf{body}}$
```

```
SimVerDec($g, u, n, c, ad$)  // $ad = \varepsilon$
require $\mathsf{K}[g] \neq \perp$ and $\neg\mathsf{group\_is\_corrupt}[g]$
require $\mathsf{vk}[u] \neq \perp$ and $u \in \mathsf{members}[g]$
$m_e \leftarrow \text{Dec}(g, n, c)$ ; If $m_e = \perp$ then return $\perp$
abort(1)

SIGENC-DER[NE, H, 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 $m_e$

ENC-DER[$m$]($K$)
Return $m$

SimNewHonUser($u$)
SimExposeUser($u$)
SimNewCorrUser($u, sk, vk$)
// These oracles are identical to the corresponding
// oracles in the OAE game for SP as per Fig. 7.
```

```
SimNewHonGroup($g$, users)
require $\mathsf{K}[g] = \perp$
$\mathsf{K}[g] \leftarrow$ "dummy"
NewHonGroup($g$)
$\mathsf{members}[g] \leftarrow \mathsf{users}$

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

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

Fig. 36: **Top pane:** Games $\mathcal{G}_0$–$\mathcal{G}_3$ for the proof of Theorem 5. The code highlighted in gray was added by expanding the symmetric signcryption algorithms SP.SigEnc, SP.VerDec, the ciphertext-triviality predicate $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$, and the output-guarding function $\mathsf{func}_{\mathsf{out}}^{\perp}$ in game $\mathcal{G}_{\mathsf{SP},\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}},\mathsf{func}_{\mathsf{out}}^{\perp}}^{\mathsf{OAE}[\text{Enc}]}(\mathcal{A}_{\mathsf{IUF\text{-}of\text{-}SP}})$. The code added for the transitions between games is highlighted in green. **Bottom pane:** Adversary $\mathcal{A}_{\mathsf{KDMAE}}$ for the proof of Theorem 5. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_3$.

Transition from $\mathcal{G}_1$ to $\mathcal{G}_2$. In game $\mathcal{G}_2$ we modify VerDec to return $\bot$ whenever bad_0 is set. As the games $\mathcal{G}_1$ and $\mathcal{G}_2$ are identical-until-bad, $\Pr[\mathcal{G}_1] \leq \Pr[\mathcal{G}_2] + \Pr[\mathsf{bad}_0^{\mathcal{G}_1}]$.

Bounding $\Pr[\mathsf{bad}_0^{\mathcal{G}_1}]$. We bound the probability that bad_0 is set in $\mathcal{G}_1$ by describing adversary $\mathcal{A}_{\mathsf{SPARSE}}$ that perfectly simulates $\mathcal{G}_1$ for $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$. Adversary $\mathcal{A}_{\mathsf{SPARSE}}$ plays in the SPARSE security game against DS with respect to the message set $\mathcal{M}$. It simply runs the code for the game $\mathcal{G}_1$ by sampling and maintaining the user and group keys. Whenever the flag bad_0 is set on a VerDec query, adversary $\mathcal{A}_{\mathsf{SPARSE}}$ returns the corresponding $(\mathsf{vk}[u], m_s, s)$ and wins its game. Note that the bad_0 flag is set when the output c of a prior Enc query is successfully unsigncrypted by VerDec. The correctness of NE guarantees that $m_e = \mathsf{NE.Dec}(\mathsf{K}[g], n, c)$ is the same as the message previously encrypted by Enc. As Enc requires that $m_e \in \mathcal{M}$, we have $\mathsf{DS.Ver}(\mathsf{vk}[u], m_s, s) = \mathsf{true}$ and $s \,\|\, m_e \in \mathcal{M}$. These are the same as the winning conditions in the SPARSE game of DS. Thus

$$\Pr[\mathsf{bad}_0^{\mathcal{G}_1}] \leq \mathsf{Adv}^{\mathsf{SPARSE}}_{\mathsf{DS},\mathcal{M}}(\mathcal{A}_{\mathsf{SPARSE}}).$$

Transition from $\mathcal{G}_2$ to $\mathcal{G}_3$. In game $\mathcal{G}_3$, we modify the VerDec oracle. We move the instruction that checks if $(g, n, c) \in L$ to the top of the oracle. This does not change the output of the oracle and we have

$$\Pr[\mathcal{G}_2] = \Pr[\mathcal{G}_3].$$

Bounding $\Pr[\mathcal{G}_3]$. In Fig. 36, we define the adversary $\mathcal{A}_{\mathsf{KDMAE}}$ that simulates $\mathcal{G}_3$ for $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$. The reduction simulates responses to group oracle queries by using its own group oracles and simulates the user oracles by sampling and maintaining keys locally. While simulating the SigEnc oracle, it replaces calls to $\mathsf{NE.Enc}$ with its oracle Enc using the function $\mathsf{SIGENC\text{-}DER}$. The function $\mathsf{SIGENC\text{-}DER}$ executes exactly the same lines of code that computed m_e from these parameters in $\mathcal{G}_3$. This means that

$$\begin{aligned}\textsc{Enc}(g, n, \mathsf{SIGENC\text{-}DER}[\mathsf{NE}, \mathsf{H}, \mathsf{DS}, m_0, n, \mathsf{sk}[u]](K), \mathsf{SIGENC\text{-}DER}[\mathsf{NE}, \mathsf{H}, \mathsf{DS}, m_1, n, \mathsf{sk}[u]](K))\\ = \mathsf{NE.Enc}(K, n, m_e).\end{aligned}$$

To simulate the encryption oracle Enc, the adversary $\mathcal{A}_{\mathsf{KDMAE}}$ uses its oracle Enc with the function $\mathsf{ENC\text{-}DER}$. The function $\mathsf{ENC\text{-}DER}$ is parameterized by a message m and returns m. Note that the output of $\textsc{Enc}(g, n_{\mathsf{body}}, \mathsf{ENC\text{-}DER}[m_{\mathsf{body},0}](K), \mathsf{ENC\text{-}DER}[m_{\mathsf{body},1}](K))$ is the same as $\mathsf{NE.Enc}(K, n_{\mathsf{body}}, m_{\mathsf{body},b})$. While simulating the VerDec oracle, the reduction replaces calls to $\mathsf{NE.Dec}$ with its Dec oracle. Adversary $\mathcal{A}_{\mathsf{AEAD}}$ wins whenever $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ does. Therefore,

$$\Pr[\mathcal{G}_3] \leq \mathsf{Adv}^{\mathsf{KDMAE}}_{\mathsf{NE},\Phi_{\mathsf{SP}}}(\mathcal{A}_{\mathsf{AEAD}}).$$

D Proofs for the Stronger Out-Group AE Security Results

D.1 Proof for the UV Security of StE (Proof of Proposition 2)

Proof. This proof uses the game $\mathcal{G}$ in Fig. 37. We establish that $\mathsf{Adv}^{\mathsf{UV}}_{\mathsf{StE}}(\mathcal{A}^{\mathsf{StE}}_{\mathsf{UV}}) \leq \mathsf{Adv}^{\mathsf{UV}}_{\mathsf{DS}}(\mathcal{A}^{\mathsf{DS}}_{\mathsf{UV}})$. Game $\mathcal{G}_0$ is equivalent to the game $\mathcal{G}^{\mathsf{UV}}_{\mathsf{StE}}$. The former expands multiple instructions in the latter; the expanded code is marked in gray. Therefore, $\mathsf{Adv}^{\mathsf{UV}}_{\mathsf{StE}}(\mathcal{A}^{\mathsf{StE}}_{\mathsf{UV}}) = \Pr[\mathcal{G}_0]$.

In Fig. 37 we show adversary $\mathcal{A}^{\mathsf{DS}}_{\mathsf{UV}}$ against the UV security of DS that simulates the game $\mathcal{G}_0$ for $\mathcal{A}^{\mathsf{StE}}_{\mathsf{UV}}$ and wins whenever $\mathcal{A}^{\mathsf{StE}}_{\mathsf{UV}}$ does. Once $\mathcal{A}^{\mathsf{StE}}_{\mathsf{UV}}$ returns $(g, K, u', u, sk_{u'}, vk_u, n, m, ad)$, the reduction constructs $m_s = \langle g, n, m, ad \rangle$, and returns $(sk_{u'}, vk_{u'}, m_s)$. Note that whenever $\mathcal{A}^{\mathsf{StE}}_{\mathsf{UV}}$ wins, it must be the case that $\mathsf{SS.VerDec}(g, K, u', vk_{u'}, n, c, ad) \neq \bot$ and $((sk_u, vk_{u'}) \notin [\mathsf{SS.UserKg}])$. By the correctness of NE and the construction of StE, this implies that $\mathsf{DS.Ver}(vk_{u'}, m_s, s)$ and $((sk_u, vk_{u'}) \notin [\mathsf{SS.UserKg}])$ where $m_s = \langle g, n, m, ad \rangle$. These two are exactly the winning conditions of the UV security game of DS. Therefore, $\mathcal{A}^{\mathsf{DS}}_{\mathsf{UV}}$ wins whenever $\mathcal{A}^{\mathsf{StE}}_{\mathsf{UV}}$ does and it follows that $\mathsf{Adv}^{\mathsf{UV}}_{\mathsf{DS}}(\mathcal{A}^{\mathsf{DS}}_{\mathsf{UV}}) \geq \Pr[\mathcal{G}_0]$.

For simplicity, we only give the proof of Proposition 2 for $\mathsf{SS} = \mathsf{StE}$. However, the adversary $\mathcal{A}^{\mathsf{DS}}_{\mathsf{UV}}$ of Fig. 37 remains the same for all $\mathsf{SS} \in \{\mathsf{StE}, \mathsf{EtS}, \mathsf{SP}\}$ except for the line in the $\boxed{\text{box}}$ that constructs the value m_s. For other choices of SS, this line would be replaced as per the construction of the scheme.

```
$\mathcal{G}(\mathcal{A}_{\mathsf{UV}}^{\mathsf{StE}})$
$(g, K, u, u', sk_u, vk_{u'}, n, m, ad) \leftarrow\$ \mathcal{A}_{\mathsf{UV}}^{\mathrm{U,G,SigEnc,VerDec}}$
 $//\ c \leftarrow \mathsf{StE.SigEnc}(g, K, u, sk_u, n, m, ad)$
$m_s \leftarrow \langle g, n, m, ad \rangle$
$s \leftarrow \mathsf{DS.Sig}(sk_u, m_s)$
$m_e \leftarrow s \,\|\, m\,;\ ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$
$c \leftarrow \mathsf{NE.Enc}(K_g, n, m_e, ad_{\mathsf{NE}})$
 $//\ \mathsf{win}_0 \leftarrow (\mathsf{StE.VerDec}(g, K, u', vk_{u'}, n, c, ad) \neq \bot)$
$ad_{\mathsf{NE}} \leftarrow \langle u', ad \rangle$
$m_e \leftarrow \mathsf{NE.Dec}(K_g, n, c, ad_{\mathsf{NE}})$
If $m_e = \bot$ then return $\mathsf{win}_0 \leftarrow \mathsf{false}$
Else
   $s \,\|\, m \leftarrow m_e$   $//$ s.t. $|s| = \mathsf{DS.sl},\ |m| \geq 0$
   $m_s \leftarrow \langle g, n, m, ad \rangle$
   If $\neg\mathsf{DS.Ver}(vk_{u'}, m_s, s)$ then $\mathsf{win}_0 \leftarrow \mathsf{false}$
   Else $\mathsf{win}_0 \leftarrow \mathsf{true}$
$\mathsf{win}_1 \leftarrow ((sk_u, vk_{u'}) \notin [\mathsf{SS.UserKg}])$
Return $\mathsf{win}_0$ and $\mathsf{win}_1$
```

```
$\mathcal{A}_{\mathsf{UV}}^{\mathsf{DS}}$
$(g, K, u, u', sk_u, vk_{u'}, n, m, ad) \leftarrow\$ \mathcal{A}_{\mathsf{UV}}^{\mathrm{U,G,SigEnc,VerDec}}$
$\boxed{m_s \leftarrow \langle g, n, m, ad \rangle}$
Return $(sk_u, vk_{u'}, m_s)$
```

Fig. 37: The game $\mathcal{G}$ and the adversary $\mathcal{A}_{\mathsf{UV}}^{\mathsf{DS}}$ used in the proof of Proposition 2.

```
Games $\mathcal{G}_0$–$\mathcal{G}_2$
$(sk, vk', m) \leftarrow\$ \mathcal{A}_{\mathsf{UV}}^{\mathrm{H}}$
$s \leftarrow \mathsf{Ed25519.Sig}(sk, m)$
$R \,\|\, z \leftarrow s\,;\ A' \leftarrow vk'$
If $z \geq p$ then return $\bot$
If $R \notin \mathbb{G}_p$ or $A' \notin \mathbb{G}$ then
   Return $\mathsf{false}$
$\mathsf{ch}' \leftarrow \mathrm{H}(R, A', m)$
If $z \cdot \mathbf{B} = R + \mathsf{ch}' \cdot A'$ then
   $\mathsf{win}_0 \leftarrow \mathsf{true}$
$\mathsf{win}_1 \leftarrow ((sk, vk') \notin [\mathsf{DS.Kg}])$
Return $\mathsf{win}_0$ and $\mathsf{win}_1$
```

```
$\mathrm{H}(x)$
If $\mathsf{H}[x] = \bot$ then
   $X \leftarrow\$ \mathbb{Z}_p$
   $\mathsf{H}[x] \leftarrow X$
$(R, A, m) \leftarrow x\,;\ \mathsf{ch} \leftarrow \mathsf{H}[x]$
If $\mathsf{ch} = 0$ then
   $\mathsf{bad}_0 \leftarrow \mathsf{true}$
   $\mathsf{ch} \leftarrow\$ \mathbb{Z}_p \setminus \{0\}$   $//\ \mathcal{G}_{[1,\infty)}$
   $\mathsf{H}[x] \leftarrow \mathsf{ch}$   $//\ \mathcal{G}_{[1,\infty)}$
If $\exists A' : T[(R, A', m)] = \mathsf{ch} \cdot A$ and $A \neq A'$ then   $//\ \mathsf{ch}' \cdot A' = \mathsf{ch} \cdot A$
   $\mathsf{bad}_1 \leftarrow \mathsf{true}$
   While $(\exists A' : T[(R, A', m)] = \mathsf{ch} \cdot A$ and $A \neq A')$ or $\mathsf{ch} \cdot A = 0$ do   $//\ \mathcal{G}_{[2,\infty)}$
      $\mathsf{ch} \leftarrow\$ \mathbb{Z}_p$   $//\ \mathcal{G}_{[2,\infty)}$
   $\mathsf{H}[x] \leftarrow \mathsf{ch}$   $//\ \mathcal{G}_{[2,\infty)}$
$T[(R, A, m)] = \mathsf{ch} \cdot A$
Return $\mathsf{H}[x]$
```

Fig. 38: Games $\mathcal{G}_0$–$\mathcal{G}_2$ used in the proof of Proposition 3.

D.2 Proof for the UV Security of Ed25519 (Proof of Proposition 3)

Proof. OVERVIEW. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_2$ in Fig. 38. We establish the following claims.

- $\mathsf{Adv}_{\mathsf{Ed25519}}^{\mathsf{UV}}(\mathcal{A}_{\mathsf{UV}}) = \Pr[\mathcal{G}_0]$
- $\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$
- $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq 2 \cdot q_{\mathrm{H}}/p$
- $\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \leq \Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$
- $\Pr[\mathcal{G}_2] = 0$

By combining the above, we have

$$\begin{aligned}\mathsf{Adv}_{\mathsf{Ed25519}}^{\mathsf{UV}}(\mathcal{A}_{\mathsf{UV}}) = \Pr[\mathcal{G}_0] &= \Pr[\mathcal{G}_0] = (\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1]) + \Pr[\mathcal{G}_1] \\ &\leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}] + \Pr[\mathcal{G}_1] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}] + (\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2]) + \Pr[\mathcal{G}_2] \\ &\leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}] + \Pr[\mathsf{bad}_1^{\mathcal{G}_1}] \leq (2 \cdot q_{\mathrm{H}})/p + q_{\mathrm{H}}^2/p.\end{aligned}$$

We justify our claims in the following paragraphs.

Game $\mathcal{G}_0$. The game $\mathcal{G}_0$ is functionally equivalent to $\mathcal{G}^{\mathsf{UV}}_{\mathsf{Ed25519}}$. We use gray highlighting to show code from $\mathsf{Ed25519}$. Additionally, $\mathcal{G}_0$ includes code lines highlighted in green that we use for transitions between games. In particular, the random oracle H sets the bad_0 flag if the challenge ch sampled for a query $x = (R, A, m)$ satisfies $\mathsf{ch} \cdot A = 0$. The random oracle H also maintains a dictionary T that stores the value $\mathsf{ch} \cdot A$ corresponding to every query (R, A, m) where ch is the output for that query. It sets the bad_1 flag if the booleans $\exists A' : T[(R, A', m)] = \mathsf{ch} \cdot A$ and $A \neq A'$ evaluate to true for a query (R, A, m). These changes do not affect the functionality of $\mathcal{G}_0$. It follows that

$$\Pr[\mathcal{G}_0] = \mathsf{Adv}^{\mathsf{UV}}_{\mathsf{Ed25519}}(\mathcal{A}_{\mathsf{UV}}).$$

Transitions from $\mathcal{G}_0$ to $\mathcal{G}_2$. Let $i \in \{0, 1\}$. Games $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ are identical until bad_i is set, therefore

$$\Pr[\mathcal{G}_i] - \Pr[\mathcal{G}_{i+1}] \leq \Pr[\mathsf{bad}_i^{\mathcal{G}_i}].$$

Bounding $\Pr[\mathsf{bad}_0^0]$. We now compute $\Pr[\mathsf{bad}_0^0]$. Consider the condition $\mathsf{ch} \cdot A = 0$ that sets the bad_0 flag for an input (R, A, m) in game $\mathcal{G}_0$. Recall that ch is sampled uniformly at random from $\mathbb{Z}_p$. Now for a given $A = a \cdot \mathbf{B}$ there are exactly two values for ch – $0_{\mathbf{B}}$ and $a^{-1} \bmod p$ – that satisfy $\mathsf{ch} \cdot A = 0$. Therefore the probability that a particular random oracle query sets the bad_0 flag in $\mathcal{G}_0$ is $2/p$. Applying a union bound over all H queries we have

$$\Pr[\mathsf{bad}_0^0] = q_{\mathrm{H}} \cdot 2/p.$$

Bounding $\Pr[\mathsf{bad}_1^1]$. Consider the condition that sets bad_1 in $\mathcal{G}_1$. We have $T(R, A', m) = \mathsf{ch} \cdot A$ and $A' \neq A$. Let ch' be the response returned for the query (R, A', m). Then we have $\mathsf{ch}' \cdot A' = \mathsf{ch} \cdot A$. At the time bad_1 is set, ch', A', and A are fixed. The probability that for a particular random oracle query $\mathsf{ch} = (\mathsf{ch}' \cdot A') \cdot A^{-1}$ is bounded by q_{H}/p. Applying a union bound over all random oracle queries we have

$$\Pr[\mathsf{bad}_1^1] = q_{\mathrm{H}}^2/(p-1).$$

Game $\mathcal{G}_2$. Consider the winning conditions in $\mathcal{G}_2$. The adversary $\mathcal{A}_{\mathsf{UV}}$ wins the game $\mathcal{G}_2$ if it returns (sk, vk', m) such that $(z \cdot \mathbf{B} = R + \mathsf{ch}' \cdot vk')$ and $((sk, vk') \notin [\mathsf{DS.Kg}])$ where $(R \,\|\, z) = \mathsf{Ed25519.Sig}(sk, m)$ and ch'. Let vk be the verification key that satisfies $(sk, vk) \in [\mathsf{DS.Kg}]$. Then we have from the correctness of DS that $z \cdot \mathbf{B} = R + \mathsf{ch}' \cdot vk$. We also know that $vk \neq vk'$. So we have $z \cdot \mathbf{B} = R + \mathsf{ch}' \cdot vk'$ and $z \cdot \mathbf{B} = R + \mathsf{ch}' \cdot vk$. These two conditions can simultaneously hold only if one of $\mathsf{ch} \cdot A = 0$ or $\mathsf{ch}' \cdot A' = \mathsf{ch} \cdot A$ is true. In our transitions from $\mathcal{G}_0$ to $\mathcal{G}_2$, we precluded either of these conditions from being true. Therefore, it is impossible for $\mathcal{A}_{\mathsf{UV}}$ to win $\mathcal{G}_2$. It follows that

$$\Pr[\mathcal{G}_0] = 0.$$

D.3 Stronger Out-Group AE Security Results (Proof of Proposition 4)

Overview. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_2$ in Fig. 39. We establish the following claims.

- $\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{OAE}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1$
- $\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$
- $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq \mathsf{Adv}^{\mathsf{UV}}_{\mathsf{SS}}(\mathcal{A}_{\mathsf{UV}})$
- $\Pr[\mathcal{G}_1] = \Pr[\mathcal{G}_2]$
- $2 \cdot \Pr[\mathcal{G}_2] - 1 \leq \mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}_{\mathsf{trivial}},\mathsf{func}_{\mathsf{out}}\bot}(\mathcal{A}_{\mathsf{OAE}\bot})$

By combining the above, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}_{\mathsf{trivial}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{OAE}}) &= 2 \cdot \Pr[\mathcal{G}_0] - 1 = 2 \cdot ((\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1]) + \Pr[\mathcal{G}_1]) - 1 \\ &\leq 2 \cdot (\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] + \Pr[\mathcal{G}_1]) - 1 \\ &\leq \mathsf{Adv}^{\mathsf{UV}}_{\mathsf{SS}}(\mathcal{A}_{\mathsf{UV}}) + \mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}_{\mathsf{trivial}},\mathsf{func}_{\mathsf{out}}\bot}(\mathcal{A}_{\mathsf{OAE}\bot}).\end{aligned}$$

We justify our claims in the following paragraphs.

```
Games $\mathcal{G}_0$–$\mathcal{G}_2$
$b \leftarrow_{\$} \{0,1\}$ ; $b' \leftarrow_{\$} \mathcal{A}_{\mathsf{OAE}}^{\mathrm{U,G,SigEnc,VerDec}}$
Return $b = b'$

NewHonUser($u$)
NewHonGroup($g$, users)
ExposeUser($u$)
ExposeGroup($g$)
NewCorrUser($u, sk, vk$)
NewCorrGroup($g, K$, users)
// These oracles are identical to the corresponding
// oracles in the OAE game for StE as per Fig. 7.

Nonce reuse checks
require $\forall d \in \{0,1\}, (g,u,n,m_d,ad) \notin N_d$   // SS ∈ {EtS, StE}
require $(g,n) \notin N$   // SS = SP
$(n_{\mathsf{body}}, n_{\mathsf{header}}) \leftarrow n$   // SS = BM
If $n_{\mathsf{body}} = n_{\mathsf{header}}$ then return $\bot$   // SS = BM
If $(g, n_{\mathsf{body}}) \in N$ then return $\bot$   // SS = BM
If $(g, n_{\mathsf{header}}) \in N$ then return $\bot$   // SS = BM

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 $u \in \mathsf{members}[g]$
  // Nonce reuse check
If $m_0 \neq m_1$ then
    If $\mathsf{group\_is\_corrupt}[g]$ then return $\bot$
    Else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$c \leftarrow \mathsf{SS.SigEnc}(g, \mathsf{K}[g], u, \mathsf{sk}[u], n, m, ad)$
  // Update nonce set
$C \leftarrow C \cup \{((g,u,n,m_b,ad),c)\}$
$Q \leftarrow Q \cup \{((g,u,n,m_0,m_1,ad),c)\}$
$W[g,u,n,c,ad] \leftarrow m_1$ ; Return $c$

Update nonce set
$N_0 \leftarrow N_0 \cup \{(g,u,n,m_0,ad)\}$   // SS ∈ {EtS, StE}
$N_1 \leftarrow N_1 \cup \{(g,u,n,m_1,ad)\}$   // SS ∈ {EtS, StE}
$N \leftarrow N \cup \{(g,n)\}$   // SS = SP
$N \leftarrow N \cup \{(g, n_{\mathsf{header}}), (g, n_{\mathsf{body}})\}$   // SS = BM
```

```
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 $u \in \mathsf{members}[g]$
If $W[g,u,n,c,ad] \neq \bot$ then   // $\mathcal{G}_{[2,\infty)}$
    If $(\mathsf{vk}[u], \mathsf{sk}[u]) \notin [\mathsf{SS.UserKg}]$ then   // $\mathcal{G}_{[2,\infty)}$
        Return $\bot$   // $\mathcal{G}_{[2,\infty)}$
    Return $W[g,u,n,c,ad]$   // $\mathcal{G}_{[2,\infty)}$
$m \leftarrow \mathsf{SS.VerDec}(g, \mathsf{K}[g], u, \mathsf{sk}[u], n, c, ad)$
If $m = \bot$ then return $\bot$
$z \leftarrow ((g,u,n,m,ad),c)$
If $z \in C$ then
    If $(\mathsf{vk}[u], \mathsf{sk}[u]) \notin [\mathsf{SS.UserKg}]$ then
        $\mathsf{bad}_0 \leftarrow \mathsf{true}$
        Return $\bot$   // $\mathcal{G}_{[1,2)}$
    Return $\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}(z, Q)$
If $b = 0$ then return $\bot$ else return $m$
```

```
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 $u \in \mathsf{members}[g]$
If $\exists u' : W[g,u',n,c,ad] \neq \bot$ then   // $\mathcal{G}_{[2,\infty)}$
    If $(\mathsf{vk}[u], \mathsf{sk}[u']) \notin [\mathsf{SS.UserKg}]$ then   // $\mathcal{G}_{[2,\infty)}$
        Return $\bot$   // $\mathcal{G}_{[2,\infty)}$
    Return $W[g,u',n,c,ad]$   // $\mathcal{G}_{[2,\infty)}$
$m \leftarrow \mathsf{SS.VerDec}(g, \mathsf{K}[g], u, \mathsf{sk}[u], n, c, ad)$
If $m = \bot$ then return $\bot$
$z \leftarrow ((g,u,n,m,ad),c)$
If $\exists u' : ((g,u',n,m,ad),c) \in C$ then
    If $(\mathsf{vk}[u], \mathsf{sk}[u']) \notin [\mathsf{SS.UserKg}]$ then
        $\mathsf{bad}_0 \leftarrow \mathsf{true}$
        Return $\bot$   // $\mathcal{G}_{[1,2)}$
    Return $\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}(z, Q)$
If $b = 0$ then return $\bot$ else return $m$
```

Fig. 39: Games $\mathcal{G}_0$–$\mathcal{G}_2$ used in the proof of Proposition 4 for $\mathsf{pred}_{\mathsf{trivial}} = \mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$ and $\mathsf{pred}_{\mathsf{trivial}} = \mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$. The VerDec oracle correspodning to $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$ is shown on the right and the VerDec oracle corresponding to $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ is shown on the right.

Game $\mathcal{G}_0$. The game $\mathcal{G}_0$ is functionally equivalent to $\mathcal{G}^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}_{\mathsf{trivial}},\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}}$. Additionally, the former includes code lines highlighted in green that we use for transitions between games. The VerDec oracle corresponding to $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$ is shown at the bottom-left of Fig. 39 and that corresponding to $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ is shown at the bottom-right of Fig. 39. The Derive_vk function defined in ?? derives the verification key vk corresponding to the input signing key sk. The conditional statement that was added in VerDec oracle sets the bad_0 flag whenever the boolean $(\mathsf{vk}[u], \mathsf{sk}[u]) \notin [\mathsf{SS.UserKg}]$ evaluates to true for $\mathsf{pred}_{\mathsf{trivial}} = \mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$ and whenever the boolean $(\mathsf{vk}[u], \mathsf{sk}[u]) \notin [\mathsf{SS.UserKg}]$ evaluates to true for $\mathsf{pred}_{\mathsf{trivial}} = \mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$. These changes do not affect the functionality of $\mathcal{G}_0$ and we have

$$\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}_{\mathsf{trivial}},\mathsf{func}_{\mathsf{out}}^{\mathsf{sec}}}(\mathcal{A}_{\mathsf{OAE}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1.$$

Transition from $\mathcal{G}_0$ to $\mathcal{G}_1$. In game $\mathcal{G}_1$ we modify the VerDec oracle such that it returns $\bot$ whenever bad_0 is set. As the games $\mathcal{G}_0$ and $\mathcal{G}_1$ are identical-until-bad, we have

$$\Pr[\mathcal{G}_0] \leq \Pr[\mathcal{G}_1] + \Pr[\mathsf{bad}_0^{\mathcal{G}_0}].$$

Bounding $\Pr[\mathsf{bad}_0^0]$. In order to bound $\Pr[\mathsf{bad}_0^0]$, consider an adversary $\mathcal{A}_{\mathsf{UV}}$ against the UV security of SS that is defined as follows. It perfectly simulates the game $\mathcal{G}_0$ for adversary $\mathcal{A}_{\mathsf{OAE}}$ until the bad_0^0 flag is set. To simulate the user oracles U and the group oracles G, adversary $\mathcal{A}_{\mathsf{UV}}$ itself samples and maintains all user keys and group keys.

Let us first define $\mathcal{A}_{\mathsf{UV}}$ for the case $\mathsf{pred}_{\mathsf{trivial}} = \mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$. Whenever the bad_0 flag is set in the simulated game during a VerDec query, adversary $\mathcal{A}_{\mathsf{UV}}$ returns the corresponding $(g, \mathsf{K}[g], u, u, \mathsf{sk}[u], \mathsf{vk}[u], n, m, ad)$. Consider a VerDec query (g, u, n, c, ad) that sets the bad_0 flag in $\mathcal{G}_0$. During this query, the verification key $\mathsf{vk}[u]$ successfully verifies and decrypts the ciphertext c, i.e., $\mathsf{SS.VerDec}(g, \mathsf{K}[g], u, \mathsf{vk}[u], n, c, ad) \neq \bot$. Moreover, we have $z = (g, u, n, m, ad) \in C$. Therefore, $c = \mathsf{SS.SigEnc}(g, \mathsf{K}[g], u, \mathsf{sk}[u], n, c, ad)$. Note that immediately before the bad_0 flag is set, game $\mathcal{G}_0$ verifies that $(\mathsf{vk}[u], \mathsf{sk}[u]) \notin [\mathsf{SS.UserKg}]$ holds. This implies that $(\mathsf{sk}[u], \mathsf{vk}[u]) \notin [\mathsf{DS.Kg}]$. So the bad_0 flag is set when the following conditions are true – $\mathsf{SS.VerDec}(g, \mathsf{K}[g], u, \mathsf{vk}[u], n, c, ad) \neq \bot$ where $c = \mathsf{SS.SigEnc}(g, \mathsf{K}[g], u', \mathsf{sk}[u'], n, c, ad)$ and $(\mathsf{sk}[u], \mathsf{vk}[u]) \notin [\mathsf{DS.Kg}]$.

We now define $\mathcal{A}_{\mathsf{UV}}$ for the case $\mathsf{pred}_{\mathsf{trivial}} = \mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$. Whenever the bad_0 flag is set in the simulated game during a VerDec query, adversary $\mathcal{A}_{\mathsf{UV}}$ returns the corresponding $(g, \mathsf{K}[g], u', u\mathsf{sk}[u'], \mathsf{vk}[u], n, m, ad)$. Consider a VerDec query (g, u, n, c, ad) that sets the bad_0 flag in $\mathcal{G}_0$. During this query, the verification key $\mathsf{vk}[u]$ successfully verifies and decrypts the ciphertext c, i.e., $\mathsf{SS.VerDec}(g, \mathsf{K}[g], u, \mathsf{vk}[u], n, c, ad) \neq \bot$. Moreover we have $\exists u' : ((g, u', n, c, ad), c) \in C$. Therefore, $c = \mathsf{SS.SigEnc}(g, \mathsf{K}[g], u', \mathsf{sk}[u'], n, c, ad)$. Note that immediately before the bad_0 flag is set, game $\mathcal{G}_0$ verifies that $(\mathsf{vk}[u], \mathsf{sk}[u']) \notin [\mathsf{SS.UserKg}]$ holds. This implies that $(\mathsf{sk}[u'], \mathsf{vk}[u]) \notin [\mathsf{DS.Kg}]$. So the bad_0 flag is set when the following conditions are true – $\mathsf{SS.VerDec}(g, \mathsf{K}[g], u, \mathsf{vk}[u], n, c, ad) \neq \bot$ where $c = \mathsf{SS.SigEnc}(g, \mathsf{K}[g], u', \mathsf{sk}[u'], n, c, ad)$ and $(\mathsf{sk}[u'], \mathsf{vk}[u]) \notin [\mathsf{DS.Kg}]$.

In both cases, the conditions under which bad_0 is set in $\mathcal{G}_0$ are exactly the winning conditions in the UV game of SS. Therefore, by returning the corresponding values, $\mathcal{A}_{\mathsf{UV}}$ wins its game and we have

$$\Pr[\mathsf{bad}_0^0] \leq \mathsf{Adv}_{\mathsf{SS}}^{\mathsf{UV}}(\mathcal{A}_{\mathsf{UV}}).$$

Transition from $\mathcal{G}_1$ to $\mathcal{G}_2$. First, we address this transition for the case $\mathsf{pred}_{\mathsf{trivial}} = \mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$. In the game $\mathcal{G}_1$, if the adversary forwards an honestly generated ciphertext from SigEnc to VerDec, then $z \in C$ evaluates to true in VerDec and VerDec returns the corresponding challenge message m_1 if $(\mathsf{vk}[u], \mathsf{sk}[u]) \in [\mathsf{SS.UserKg}]$ and $\bot$ otherwise. In game $\mathcal{G}_2$ we remove the conditional that checks if $\mathsf{pred}_{\mathsf{trivial}}(z, C)$ is true from VerDec and instead use the table W to achieve the same behavior without actually having to decrypt forwarded ciphertexts. For every SigEnc query, the table W indexed by g, u, n, c, and ad stores the challenge message m_1 where c is the ciphertext returned for that query. This table W is then used to respond to VerDec queries on trivially forwarded ciphertexts. Note that the decryption correctness of SS gives $W[g, u, n, c, ad] \neq \bot$ evaluates to true in $\mathcal{G}_2$ iff $\mathsf{pred}_{\mathsf{trivial}}(z, C)$ for $z = ((g, u, n, m, ad), c)$. Therefore the outputs of VerDec are consistent between the two games.

Now we analyze the case $\mathsf{pred}_{\mathsf{trivial}} = \mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ analogously. In the game $\mathcal{G}_1$, if the adversary forwards an honestly generated ciphertext from SigEnc to VerDec, then $\exists u' : ((g, u', n, m, ad), c) \in C$ evaluates to true in VerDec and VerDec returns the corresponding challenge message m_1 if $(\mathsf{vk}[u], \mathsf{sk}[u']) \in [\mathsf{SS.UserKg}]$ and $\bot$ otherwise. Again we use the table W to replace the conditional that checks the triviality predicate. The decryption correctness of SS gives $\exists u' : W[g, u', n, c, ad] \neq \bot$ evaluates to true in $\mathcal{G}_2$ iff $\exists u' : ((g, u', n, m, ad), c) \in C$. So the outputs of VerDec are consistent between the two games.

In both cases, the outputs of VerDec remain unchanged between $\mathcal{G}_1$ and $\mathcal{G}_2$. It follows that

$$\Pr[\mathcal{G}_1] = \Pr[\mathcal{G}_2].$$

Bounding $\Pr[\mathcal{G}_2]$. In Fig. 40, we define the adversary $\mathcal{A}_{\mathsf{OAE}\bot}$ against the OAE security of SS with respect to the predicate $\mathsf{pred}_{\mathsf{trivial}}$ and the output guarding function $\mathsf{func}_{\mathsf{out}}^{\bot}$. It perfectly simulates the game $\mathcal{G}_2$ for adversary $\mathcal{A}_{\mathsf{OAE}}$. In particular, it simulates the responses to the user and group oracles using the

```
Adversary $\mathcal{A}_{\mathsf{OAE}\bot}^{\mathrm{U,G,SigEnc,VerDec,Enc}}$
    $b' \leftarrow_{\$} \mathcal{A}_{\mathsf{OAE}}^{\mathrm{SimU,SimG,SimSigEnc,SimVerDec}}$
    Return $b'$
```

```
SimSigEnc$(g, u, n, m_0, m_1, ad)$
    require $\mathsf{K}[g] \neq \bot$ and $|m_{\mathsf{body},0}| = |m_{\mathsf{body},1}|$
    require $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$ and $(g, n_{\mathsf{body}}) \notin N$
    If $m_0 \neq m_1$ then
        If $\mathsf{group\_is\_corrupt}[g]$ then return $\bot$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
    $c \leftarrow \mathrm{SigEnc}(g, u, n, m_0, m_1, ad)$
    $N \leftarrow N \cup \{(g, n)\}$ ; $W[g, u, n, c, ad] \leftarrow m_1$ ; Return $c$
```

```
SimNewHonGroup$(g, \mathsf{users})$
    require $\mathsf{K}[g] = \bot$
    $\mathsf{K}[g] \leftarrow$ “dummy”
    NewHonGroup$(g)$
    $\mathsf{members}[g] \leftarrow \mathsf{users}$
```

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

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

```
SimNewHonUser$(u)$
    require $\mathsf{sk}[u] = \mathsf{vk}[u] = \bot$
    $\mathsf{vk}[u] \leftarrow$ NewHonUser$(u)$
    $\mathsf{sk}[u] \leftarrow$ ExposeUser$(u)$ ; Return $\mathsf{vk}[u]$
```

```
SimExposeUser$(u)$
    require $\mathsf{sk}[u] \neq \bot$
    Return $\mathsf{sk}[u]$
```

```
SimNewCorrUser$(u, sk, vk)$
    require $\mathsf{sk}[u] = \mathsf{vk}[u] = \bot$
    NewCorrUser$(u, sk, vk)$
    $\mathsf{sk}[u] \leftarrow sk$ ; $\mathsf{vk}[u] \leftarrow vk$
```

```
SimVerDec$(g, u, n, c, ad)$
    require $\mathsf{K}[g] \neq \bot$ and $\neg\mathsf{group\_is\_corrupt}[g]$
    require $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
    If $W[g, u, n, c, ad] \neq \bot$ then
        If $(\mathsf{vk}[u], \mathsf{sk}[u]) \notin [\mathsf{SS.UserKg}]$ then return $\bot$
        Else return $W[g, u, n, c, ad]$
    $m \leftarrow$ VerDec$(g, u, n, c, ad)$
    If $m \neq \bot$ then abort$(1)$
    Return $\bot$
```

```
SimVerDec$(g, u, n, c, ad)$
    require $\mathsf{K}[g] \neq \bot$ and $\neg\mathsf{group\_is\_corrupt}[g]$
    require $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
    If $\exists u' : W[g, u', n, c, ad] \neq \bot$ then
        If $(\mathsf{vk}[u], \mathsf{sk}[u']) \notin [\mathsf{SS.UserKg}]$ then return $\bot$
        Else return $W[g, u', n, c, ad]$
    $m \leftarrow$ VerDec$(g, u, n, c, ad)$
    If $m \neq \bot$ then abort$(1)$
    Return $\bot$
```

Fig. 40: Adversary $\mathcal{A}_{\mathsf{OAE}\bot}$ used in the proof of Proposition 4. The SimVerDec oracle for $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$ is shown on the left and the SimVerDec oracle for $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ is shown on the right.

corresponding oracles from its own OAE security game. Note that the OAE security game allows to expose all user keys, so every time an honest user key-pair is created in the simulated game $\mathcal{G}_2$, adversary $\mathcal{A}_{\mathsf{OAE}\perp}$ immediately exposes it. It simulates the SigEnc oracle using the corresponding oracle from its security game and simulates the VerDec oracle using the corresponding oracle from its security game and the table W. The simulated VerDec oracle corresponding to $\mathsf{pred}_{\mathsf{trivial}} = \mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf}}$ and $\mathsf{pred}_{\mathsf{trivial}} = \mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ are shown at the bottom-left and bottom-right respectively of Fig. 40. Finally, if the $m \neq \perp$ condition is never true in the simulated VerDec and $\mathcal{A}_{\mathsf{OAE}}$ returns b', adversary $\mathcal{A}_{\mathsf{OAE}\perp}$ also returns b'. Now consider the case that $m \neq \perp$ evaluates to true in the simulated game. It must be the case that $\mathsf{pred}_{\mathsf{trivial}}(z, C) = \mathsf{false}$ and $m \neq \perp$. This is only possible when $\mathcal{A}_{\mathsf{OAE}}$ has successfully forged a ciphertext and the bit b is 1. Therefore $\mathcal{A}_{\mathsf{OAE}\perp}$ executes $\mathbf{abort}(1)$ and wins its game. We have

$$2 \cdot \Pr[\mathcal{G}_1] - 1 \leq \mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}_{\mathsf{trivial}},\mathsf{func}^{\perp}_{\mathsf{out}}} \mathcal{A}_{\mathsf{OAE}\perp}.$$

E Other Uses of Symmetric Signcryption in Keybase

In this appendix, we briefly touch upon other uses of the SealPacket and BoxMessage algorithms in Keybase. We present three such instances – SealPacket as a standalone scheme, the algorithm SealWhole that is used to encrypt long strings, and the algorithm PostFileAttachment that is used to encrypt and upload file attachments to a third-party storage.

E.1 Security of SealPacket as a Standalone Encryption Scheme

In Keybase, the SealPacket subroutine is not only used within the BoxMessage construction to signencrypt the header but also as a standalone scheme. For example, it is used in isolation to signencrypt small chat attachments. Here we prove the wOAE security of SealPacket as a standalone scheme with respect to the predicate $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ and the output guarding function $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$.

Recall that we analyzed the security of SealPacket when used within BoxMessage in Theorem 5. In particular, we had reduced the $\mathsf{wOAE}[\textsc{Enc}[\mathcal{M}, \mathsf{NE}]]$ security of SealPacket with respect to the predicate $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ and the output guarding function $\mathsf{func}^{\perp}_{\mathsf{out}}$ to the $\mathcal{M}$-sparsity of the underlying digital signature scheme DS and the KDMAE security of the underlying nonce based encryption scheme NE. One could capture the wOAE security of the standalone SealPacket scheme with respect to the predicate $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ and the function $\mathsf{func}^{\perp}_{\mathsf{out}}$ as a special case of this result by restricting the adversary from making any calls to its Enc oracle. By doing so, the reduction to the SPARSE security of the digital signature scheme is no longer required. Moreover, the class of message-deriving functions considered during the reduction to the KDMAE security of the nonce-based encryption scheme would be smaller. This is because in the proof of Theorem 5 the function ENC-DER was only required to simulate the Enc oracle, which is never used when treating SealPacket as a standalone scheme, allowing its omition from the class of message-deriving functions considered.

In the following, we show that the wOAE security of the standalone SealPacket scheme with respect to the predicate $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ and the output guarding function $\mathsf{func}^{\perp}_{\mathsf{out}}$ implies its wOAE security with respect to the stronger output-guarding function $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$.

Proposition 8. *Let* $\mathsf{DS} = \mathsf{Ed25519\text{-}DS}[k, p, \mathbb{G}, \mathbb{G}_p, \mathbf{B}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3]$ *be the digital signature scheme built from some* $k, p, \mathbb{G}, \mathbb{G}_p, \mathbf{B}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$ *as specified in Construction 6. Let* $\mathsf{SP} = \mathsf{SEAL\text{-}PACKET\text{-}SS}[\mathsf{H}, \mathsf{DS}, \mathsf{NE}]$ *be the symmetric signcryption scheme built from some* H, NE*, and* DS*. Let* $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ *be the ciphertext-triviality predicate defined in Fig. 8. Let* $\mathsf{func}^{\perp}_{\mathsf{out}}$ *and* $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}} := \mathsf{func}^{\mathsf{silence\text{-}with\text{-}m_1}}_{\mathsf{out}}[\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}]$ *be the output-guarding functions as defined in Fig. 9. Let* wOAE *be the security notion as defined in Definition 1. Let* $\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ *be any adversary against the* wOAE *security of* SP *with respect to* $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ *and* $\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}$*. Then we can build adversaries* $\mathcal{A}_{\mathsf{UV}}$ *and* $\mathcal{B}_{\mathsf{wOAE\text{-}of\text{-}SP}}$ *such that*

$$\mathsf{Adv}^{\mathsf{wOAE}}_{\mathsf{SP},\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}},\mathsf{func}^{\mathsf{sec}}_{\mathsf{out}}}(\mathcal{A}_{\mathsf{wOAE\text{-}of\text{-}SP}}) \leq \mathsf{Adv}^{\mathsf{UV}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{UV}}) + \mathsf{Adv}^{\mathsf{wOAE}}_{\mathsf{SP},\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}},\mathsf{func}^{\perp}_{\mathsf{out}}}(\mathcal{B}_{\mathsf{wOAE\text{-}of\text{-}SP}}).$$

Proof. This proof is analogous to the proof of Proposition 4.

E.2 Multi-purpose Scheme SealWhole for Encrypting Long Strings

Basic Notation. For any $x \in \{0,1\}^*$ and $\ell \in \mathbb{N}$ such that $|x| \le \ell$, we write $\langle x \rangle_\ell$ to denote the bit-string of length ℓ that is built by padding x with leading zeros.

Symmetric Signcryption Syntax. In this section, we augment the definition of SS from Section 3 with a ciphertext-length function $\mathsf{SS.cl}\colon \mathbb{N} \to \mathbb{N}$ that is associated with every instance of SS. We then require that the signcryption algorithm SS.SigEnc encrypts any message $m \in \mathsf{SS.MS}$ into a ciphertext c of the corresponding (fixed) length $|c| = \mathsf{SS.cl}(|m|)$.

The SealWhole algorithm [Keyg] is a wrapper around the standalone SealPacket algorithm and is used in Keybase to encrypt large strings and attachments. We present the pseudocode for the SealWhole algorithm in Fig. 41 and give a high-level description of the algorithm. The SealWhole algorithm accepts the same inputs as the SealPacket algorithm. To encrypt a message m with a nonce n, the SealWhole.SigEnc algorithm first parses the message m into chunks m_i of length 1 MB. It then encrypts each m_i by calling the SealPacket algorithm with nonce $n_i = n \,\|\, \langle i \bmod 2^{64} \rangle_{64}$ to obtain ciphertext c_i. The SealWhole algorithm returns the concatenation of all c_i obtained above.

```
SealWhole.SigEnc$(g, K_g, u, sk_u, n, m, ad)$  // where $n \in \{0,1\}^{128}$, $ad = \varepsilon$
$m_0 \,\|\, m_1 \,\|\, ... \,\|\, m_l \leftarrow m$  // $|m_i| = 1$ MB for all $i < l$ and $0 \le |m_l| < 1$ MB.
For $i = 0$ to $l$ do
    $n_i \leftarrow n \,\|\, \langle i \bmod 2^{64} \rangle_{64}$
    $c_i \leftarrow$ SealPacket.SigEnc$(g, K_g, u, sk_u, n_i, m_i, ad)$
Return $c_0 \,\|\, c_1 \,\|\, ... \,\|\, c_l$

SealPacket.SigEnc$(g, K_g, u, sk_u, n_{\mathsf{chunk}}, m_{\mathsf{chunk}}, ad)$  // $n_{\mathsf{chunk}} \in \{0,1\}^{192}$, $ad = \varepsilon$
$h_{\mathsf{chunk}} \leftarrow$ SHA-512$(m_{\mathsf{chunk}})$
$s_{\mathsf{chunk}} \leftarrow$ Ed25519.Sig$(sk_u,$ "Keybase-Chat-2" $\|\, \langle K_g, n_{\mathsf{chunk}}, h_{\mathsf{chunk}} \rangle)$
$c_{\mathsf{chunk}} \leftarrow$ XSalsa20-Poly1305.Enc$(K_g, n_{\mathsf{chunk}}, s_{\mathsf{chunk}} \,\|\, m_{\mathsf{chunk}})$
Return $c_{\mathsf{chunk}}$
```

Fig. 41: The SealWhole algorithm used in Keybase for encrypting long strings. We reproduce the SealPacket algorithm from Section 5 for convenience.

E.3 Encryption of Attachments with PostFileAttachment

The PostFileAttachment [Keyh] algorithm is used to encrypt and upload files in private Keybase folders. We present the pseudocode for the PostFileAttachment algorithm in Fig. 42 and give a high-level description of the algorithm. The PostFileAttachment algorithm accepts the same inputs as the BoxMessage algorithm, except the message m_{attach} is a file that a sender wants to upload to a third-party storage, and the associated data ad is a pair $(ad_{\mathsf{attach}}, ad_{\mathsf{wrap}})$. To encrypt the file attachment m_{attach}, the PostFileAttachment algorithm first samples a 256-bit attachment encryption key k_{attach} and generates an ephemeral signing key pair $(sk_{\mathsf{attach}}, vk_{\mathsf{attach}})$ by calling SealWhole.UserKg. It then computes the nonce n_{attach} as the first 128 bits of SHA-256(ad_{attach}). Next it calls the SealWhole.SigEnc algorithm to encrypt m_{attach} using the key k_{attach}, nonce n_{attach}, and signing key sk_u to obtain the ciphertext c_{attach}. Subsequently it calls SHA-256 on m_{attach} and c_{attach} to produce the hashes h_m and h_c. It then builds the wrapper message $m_{\mathsf{wrap}} = (k_{\mathsf{attach}}, vk_{\mathsf{attach}}, n_{\mathsf{attach}}, h_m, h_c)$. Finally PostFileAttachment.SigEnc invokes BoxMessage on m_{wrap}, K_g, sk_u, n_{wrap}, and ad_{wrap} to obtain the ciphertext c_{wrap}. The algorithm returns $(c_{\mathsf{wrap}}, c_{\mathsf{attach}}, n_{\mathsf{attach}})$.

We do not further analyze the SealWhole and PostFileAttachment algorithms in this work.

```
$\underline{\mathsf{PostFileAttachment.SigEnc}(g, K_g, u, sk_u, n_{\mathsf{wrap}}, m_{\mathsf{attach}}, ad)}$   $/\!/\ K_g \in \{0,1\}^{256}$
                                        $/\!/\ n_{\mathsf{wrap}} = (n_{\mathsf{wrap\text{-}body}}, n_{\mathsf{wrap\text{-}header}})\ ;\ n_{\mathsf{wrap\text{-}body}}, n_{\mathsf{wrap\text{-}header}} \in \{0,1\}^{192}$
$(ad_{\mathsf{attach}}, ad_{\mathsf{wrap}}) \leftarrow ad$
$k_{\mathsf{attach}} \xleftarrow{\$} \{0,1\}^{256}$
$(sk_{\mathsf{attach}}, vk_{\mathsf{attach}}) \xleftarrow{\$} \mathsf{SealWhole.UserKg}$      $/\!/\ \mathsf{SealWhole} = \mathsf{SEAL\text{-}WHOLE\text{-}SS}$
$n_{\mathsf{attach}} \leftarrow \mathsf{H}(ad_{\mathsf{attach}})[1..128]$      $/\!/\ \mathsf{H} = \mathsf{SHA\text{-}256}\ ;\ n_{\mathsf{attach}}$ is assigned the first 128 bits of the $\mathsf{H}$ output
$c_{\mathsf{attach}} \xleftarrow{\$} \mathsf{SealWhole.SigEnc}(g, k_{\mathsf{attach}}, u, sk_{\mathsf{attach}}, n_{\mathsf{attach}}, m_{\mathsf{attach}}, \varepsilon)$
$h_m \leftarrow \mathsf{H}(m_{\mathsf{attach}})$
$h_c \leftarrow \mathsf{H}(c_{\mathsf{attach}})$
$m_{\mathsf{wrap}} \leftarrow (k_{\mathsf{attach}}, vk_{\mathsf{attach}}, n_{\mathsf{attach}}, h_m, h_c)$
$c_{\mathsf{wrap}} \xleftarrow{\$} \mathsf{BM.SigEnc}(g, K_g, u, sk_u, n_{\mathsf{wrap}}, m_{\mathsf{wrap}}, ad_{\mathsf{wrap}})$
$c \leftarrow (c_{\mathsf{wrap}}, c_{\mathsf{attach}}, n_{\mathsf{attach}})$
Return $c$
```

Fig. 42: The PostFileAttachment algorithm used in Keybase to encrypt and upload file attachments to a third-party storage.