Analyzing Group Chat Encryption in MLS, Session, Signal, and Matrix

Joseph Jaeger and Akshaya Kumar

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/

25 March 2025

Abstract. We analyze the composition of symmetric encryption and digital signatures in secure group messaging protocols where group members share a symmetric encryption key. In particular, we analyze the chat encryption algorithms underlying MLS, Session, Signal, and Matrix using the formalism of symmetric signcryption introduced by Jaeger, Kumar, and Stepanovs (Eurocrypt 2024). We identify theoretical attacks against each of the constructions we analyze that result from the insufficient binding between the symmetric encryption scheme and the digital signature scheme. In the case of MLS and Session, these translate into practically exploitable replay and reordering attacks by a group-insider. For Signal this leads to a forgery attack by a group-outsider with access to a user's signing key, an attack previously discovered by Balbás, Collins, and Gajland (Asiacrypt 2023). In Matrix there are mitigations in the broader ecosystem that prevent exploitation. We provide formal security theorems that each of the four constructions are secure up to these attacks. Additionally, in Session we identified two attacks outside the symmetric signcryption model. The first allows a group-outsider with access to an exposed signing key to forge arbitrary messages and the second allows outsiders to replay ciphertexts.

Contents

1 Introduction 3
1.1 Our Contributions 4

2 Preliminaries 6
2.1 Standard (Multi-key) Security Notions 6

3 Symmetric Signcryption 8
3.1 Symmetric Signcryption Definitions 8
3.2 Secure Constructions and the Importance of Binding 11

4 Analysis of Sign-then-Encrypt Constructions 13
4.1 MLS 13
4.2 Session 18

5 Analysis of Encrypt-then-Sign Constructions 21
5.1 Signal 22
5.2 Matrix 25

6 Conclusions 27

A Notation and Standard Definitions 30
A.1 Notation 30
A.2 Standard primitives 30

B Proofs for MLS 32
B.1 In-group UnForgeability of MLS-StE (Proof of Theorem 1) 32
B.2 Out-group Authenticated Encryption of MLS-StE (Proof of Theorem 2) 34

C Proofs for Session 37
C.1 In-group UnForgeability of SesLG-StE (Proof of Theorem 3) 37
C.2 Out-group Authenticated Encryption of SesLG-StE (Proof of Theorem 4) 39

D Proofs for Signal 41
D.1 In-group UnForgeability of Sig-EtS (Proof of Theorem 5) 41
D.2 Out-group Authenticated Encryption of Sig-EtS (Proof of Theorem 6) 43

E Proofs for Matrix 47
E.1 In-group UnForgeability of Mat-EtS (Proof of Theorem 7) 47
E.2 Out-group Authenticated Encryption of Mat-EtS (Proof of Theorem 8) 49

1 Introduction

Secure messaging applications have been widely deployed to protect the everyday conversations of billions of users worldwide. These applications — including WhatsApp, Signal, and iMessage — use different cryptographic algorithms to provide varying privacy and authenticity guarantees. Due to their widespread usage, particularly in sensitive contexts like protests and conflicts, analyzing the security of these applications has been the focus of numerous academic works [AMPS22, ACD19, BS20, BFG+22, CGCD+17, DFG+23, FG24, RMS18]. Oftentimes, these works provide formal statements and accompanying proofs of security. However, as argued by Albrecht and Paterson [AP24], formal proofs do not necessarily guarantee absolute security: they are bounded by the assumptions and threat models chosen in a particular study. Vulnerabilities may still exist in a system that has been "proven secure".

In this work, we analyze the cryptographic cores of the group chat encryption algorithms underlying four popular group messaging applications — the Messaging Layer Security (MLS) standard [BBR+23], Session [Sesa], Signal [Sig], and Matrix [Mata]. We formally model and analyze the composition of symmetric encryption and digital signatures within these applications using the symmetric signcryption primitive of Jaeger, Kumar, and Stepanovs (JKS24) [JKS24]. Although MLS, Signal, and Matrix have been analyzed in prior works [ACDJ23, ADJ24, ACDT21, BCG23] that identified weaknesses in the respective applications and gave formal statements about their security, we discover novel flaws in these constructions. Additionally, to the best of our knowledge, we are the first to formalize and analyze Session's group chat encryption.[1] We identify a total of fourteen theoretical attacks against these applications and show that six of these attacks — against MLS, Session, and Signal — are practically exploitable by a sufficiently motivated attacker. Finally, for each implementation, we formally state and prove the security guarantees that they achieve (roughly, we prove security up to theoretical attacks).

Group messaging. Secure (group) messaging protocols can be viewed as a composition of two primitives: key agreement and chat encryption. In the simplest two party setting, the Double Ratchet [MP16] algorithm is considered the gold standard for realizing key agreement. It asynchronously establishes a continuously evolving symmetric key between two users such that each message sent in the chat is encrypted with a fresh symmetric key. The keys derived by the Double Ratchet algorithm are then used by the chat encryption primitive, which is realized using an authenticated encryption scheme. Together, they provide advanced security guarantees like forward security and post-compromise security in addition to privacy and authenticity.

In contrast to the two party setting, secure group messaging apps operate in a dynamic environment where members are continuously added or removed from the group. This poses significant challenges when scaling the two-party protocol to groups larger than two users. For example, realizing key agreement in groups by simply running the Double-Ratchet algorithm pairwise between all group members results in a quadratic communication complexity, which quickly becomes impractical for large groups. In order to improve the efficiency of group messaging apps whilst providing as many guarantees of the two-party setting as possible, protocols like the Sender Keys [Mar14] and TreeKEM [BBR18] have emerged. While these protocols improve the efficiency of end-to-end encrypted group chats, they are complex, involving many mutually interacting cryptographic components. This complicates formal analysis and increases the likelihood of implementation bugs. Given these challenges, how can we determine if a group messaging application is secure?

Analyzing group messaging protocols. As mentioned earlier, secure group messaging protocols can be broken down into the key agreement primitive and the chat encryption primitive. A comprehensive security analysis not only requires analyzing both components individually, but also their composition and the public key infrastructure that manages user identities within the protocol. Most prior work on group messaging security has focused on analyzing the key agreement primitive, as it is the more "novel" component. Due to the inherent complexities, few works attempt to rigorously analyze an application as a whole, and even fewer focus on the chat encryption primitive specifically. We bridge this gap by narrowly analyzing chat encryption in widely deployed protocols.

[1] We analyzed Session's Legacy Groups protocol which was deployed at the time of writing. It has since been replaced by the GroupsV2 protocol.

Chat encryption in groups. Chat encryption in groups does not follow directly from chat encryption in the two-party setting — another challenge with extending the two-party protocol to groups. In the two-party setting, each message is encrypted and implicitly authenticated using a shared symmetric key and an authenticated encryption scheme.

Now consider a group messaging protocol where members of a group share a symmetric key. The protocol could use the shared key along with an authenticated encryption scheme to encrypt messages from a sender to the group. While this would prevent attacks from outside the group, members could still impersonate each other. This is particularly concerning in settings where messaging apps are used to organize large-scale protests using groups so large that the government/dictator/adversary can easily infiltrate the group. In such a case, confidentiality against the adversary is unrealistic, but we would still want that a group-insider cannot impersonate another group member. A natural way to prevent group members from impersonating each other is to use digital signatures. Consequently, group messaging protocols compose symmetric encryption with digital signatures to provide privacy, integrity, and sender authenticity. These compositions are similar to those of signcryption [ADR02], the primitive that captures the joint security of public-key encryption and digital signatures. Inspired by signcryption, JKS24 introduced symmetric signcryption to jointly capture the security of symmetric encryption and digital signatures.

Symmetric signcryption. The symmetric signcryption primitive captures chat encryption in group messaging protocols in a setting where each user possesses a signing key pair, and all users in a group share a single symmetric encryption key. In addition to the shared symmetric key and the sender's signing key, the defining algorithms SigEnc and VerDec both take explicit sender and group (key) identities, nonces, and associated data as inputs. The original definition intended for the symmetric key to be long-term. However, in the protocols we analyze, the shared symmetric keys are meant to be used exactly once and then are immediately replaced with a fresh shared symmetric key. These protocols can still be modeled using symmetric signcryption if we assume that symmetric keys are independently generated and each key is used only once, allowing them to be viewed as a collection of "one-time-use" symmetric signcryption schemes.

Symmetric signcryption defines two security notions: In-group Unforgeability (IUF) and Out-group Authenticated Encryption (OAE). The former requires the unforgeability of messages sent by users whose signing keys are unknown to the adversary, even if the adversary can assign arbitrary symmetric keys to groups. The latter ensures confidentiality and integrity against an adversary that does not know the group's symmetric key even if the adversary is allowed to maliciously pick each user's signing keys. We use symmetric signcryption to analyze chat encryption in MLS, Session, Signal, and Matrix.

1.1 Our Contributions

We model the chat encryption algorithms underlying MLS, Session, Signal, and Matrix as symmetric signcryption schemes and analyze their IUF and OAE security. We summarize the results of our analysis in Tables 1 and 2. The first two columns of these tables present our main takeaways.

Our IUF analysis reveals practically exploitable insider replay attacks against MLS and Session. The attack allows a malicious group member to replay messages originally sent by an honest member, thereby impersonating them to the group.[2] Our OAE security analysis identifies two practically exploitable attacks, one against Session and the other against Signal. The attack against Session allows an external adversary to replay ciphertexts from a group member, causing other group members to interpret the replayed ciphertexts as new messages. The attack against Signal allows an external attacker to forge ciphertexts to a group by modifying honest ciphertexts created within the group. The last two columns of the tables provide explanations for why each implementation is susceptible to these attacks, which we will discuss after describing the notion of binding in symmetric signcryption.

Implicit and explicit binding. The term *binding* was first introduced in [ADR02] for (asymmetric) signcryption, which combines public-key encryption and digital signatures. It showed that naively composing (public-key) encryption with signatures does not yield secure constructions in the multi-user setting.

[2] MLS divides time into epochs that change when users are added/removed from the group or otherwise update their secrets. Our attack allows replaying within an epoch.

Application	Practical attacks	IUF security	Signature binding		
			g	n	ad
MLS	✓	weak	✗	✗	explicit
Session	✓	weak	✗	implicit	N/A
Signal	✗	strong	explicit	N/A	explicit
Matrix	✗	weak	implicit	implicit	explicit

Table 1: Results of our IUF security analysis. MLS and Session are vulnerable to insider replay attacks due to the lack of binding between the signature and the group (key) identifier.

Application	Practical attacks	OAE Security	Ctxt binding
			u
MLS	✗	weak	implicit
Session	✓*	weak	implicit
Signal	✓**	weak	implicit
Matrix	✗	weak	implicit

Table 2: Results of our OAE security analysis. Session is susceptible to outsider replay and forgery attacks that are not captured by the OAE security game. Signal is vulnerable to outsider forgery attacks due to the use of a symmetric encryption scheme that does not provide ciphertext integrity.

To make the constructions secure, it is important to bind the two components by signing the recipient's identity and encrypting the sender's identity.

Intuitively, naively composing symmetric encryption and digital signatures for symmetric signcryption would also be insecure, requiring similar countermeasures. The two secure constructions of JKS24 bind the group (key) identifier, the nonce, and the associated data to the signature, and bind the sender identifier to the symmetric ciphertext. We show that these bindings are necessary for a construction to be secure by sketching attacks that would result from the lack of proper binding. During our analysis, we check whether constructions properly bind the underlying components. Whenever a construction fails to do so, we analyze the potential real-world consequences of the corresponding attacks.

The lack of explicit binding does not always lead to a practical attack against the protocol. Sometimes, the components are implicitly bound by virtue of the inputs that the symmetric signcryption scheme is invoked on within a protocol. We say that a signature produced in a symmetric signcryption scheme implicitly binds to the group (key) identifier g, nonce n, or associated data ad if these values can be uniquely derived from the input to the signature scheme, even if they are not individually and explicitly signed. Similarly, we say that a ciphertext implicitly binds to the sender identifier u if the identifier can be uniquely derived from the inputs to the symmetric encryption scheme.

The last two columns of the Tables 1 and 2 justify the existence of practical attacks by pointing out the corresponding lack of binding. In MLS and Session, the insider replay attacks result from the lack of binding between the signature and the group (key) identifier. The outsider attacks against Session and Signal are not due to the lack of proper binding. In Session, an outsider can replay ciphertexts because every ciphertext sent in a Session group chat contains an unauthenticated timestamp which is used to detect replays. An outsider can modify the timestamp and replay the ciphertext. Additionally, an outsider who has compromised an honest member's signing keys can forge a ciphertext to a group because Session

uses public keys as group identifiers. The outsider forgery attack against Signal is due to the use of a symmetric encryption scheme that does not provide ciphertext integrity.

A positive finding regarding MLS is that the identified insider replay attack can be easily mitigated by including the message generation[3] as part of the input to the signature scheme. The current protocol specification facilitates this solution as it already requires the associated data to be signed, and the contents of the associated data are left to the discretion of individual implementations. So implementations can include the message generation in the associated data while still remaining compliant with the protocol specification.[4]

Insider Security Notions. The IUF security definition requires unforgeability of ciphertexts even against group members. As mentioned earlier, this is a practical assumption in large groups that the adversary can easily infiltrate. Moreover the MLS architecture document [BRO+24] explicitly aims to achieve privacy and authenticity in the face of malicious insiders. Yet, formal definitions for the insider security of group messaging protocols have not been considered by most of the prior works [ACDJ23, ACDT20, ACJM20, BCG23, KPPW+21]. The definitions for the insider security of the key agreement primitive underlying MLS [AJM22] take the first step in bridging this gap. Our work further motivates the need for insider security notions by presenting exploitable attacks in real-world settings.

Related work. Chat encryption in MLS, Signal, and Matrix has been formally analyzed as part of the comprehensive analyses of the respective group messaging protocols. In particular, chat encryption in MLS was studied in [ACDT21] where it was modeled as an Encrypt-then-Sign construction, whereas the current protocol specification [BBR+23] defines it as a Sign-then-Encrypt construction to hide the sender's identity from the server. Signal and Matrix were analyzed by [BCG23] as part of a broader and general analysis of group messaging applications that use the Sender Keys protocol, with the focus being on formalizing Sender Keys-like protocols rather than any specific application implementation. Matrix was also analyzed by [ADJ24] where the Device-Oriented Group Messaging model defined therein was used to show that it achieves the basic confidentiality and integrity properties. None of these works consider insider security notions.

2 Preliminaries

We provide notation, conventions, and define standard primitives in Appendix A.

2.1 Standard (Multi-key) Security Notions

We define our security games in the multi-key setting, using the notion of key-management oracles presented by JKS24. Our asymmetric games provide the adversary with a set of user oracles $\mathrm{U} = \{\textsc{NewHonUser}, \textsc{ExposeUser}, \textsc{NewCorrUser}\}$ and symmetric games provide a set of group oracles $\mathrm{G} = \{\textsc{NewHonGroup}, \textsc{ExposeGroup}, \textsc{NewCorrGroup}\}$. These oracles allow an adversary to create honest keys, expose existing honest keys, and register corrupt keys of the adversary's choice. Exposed keys are marked as corrupted after the exposure. [5] We prepend an oracle with Sim to denote its simulation in a reduction.

IND-CPA security of nonce-based encryption schemes. We define the security game $\mathcal{G}^{\mathsf{IND\text{-}CPA}}$ for a nonce-based encryption scheme NE and an adversary $\mathcal{A}_{\mathsf{IND\text{-}CPA}}$ in Fig. 1. The advantage of $\mathcal{A}_{\mathsf{IND\text{-}CPA}}$ against the IND-CPA security of NE is given by $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{IND\text{-}CPA}}) = 2 \cdot \Pr[\mathcal{G}^{\mathsf{IND\text{-}CPA}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{IND\text{-}CPA}})] - 1$.

[3] MLS uses the term *generation* to denote the index of messages sent by a user in a group during a particular epoch.

[4] Including the generation in the associated data will leak the generation to an outsider since the associated data is sent in plain. This may introduce privacy risks. For example, an outsider may be able to determine if a user is a frequent sender or not.

[5] The difference between exposing an honest user and creating a corrupt user is that in the former, the keys are sampled from the appropriate key distribution and subsequently revealed to the adversary, whereas in the latter, the adversary has the ability to maliciously select the corresponding keys.

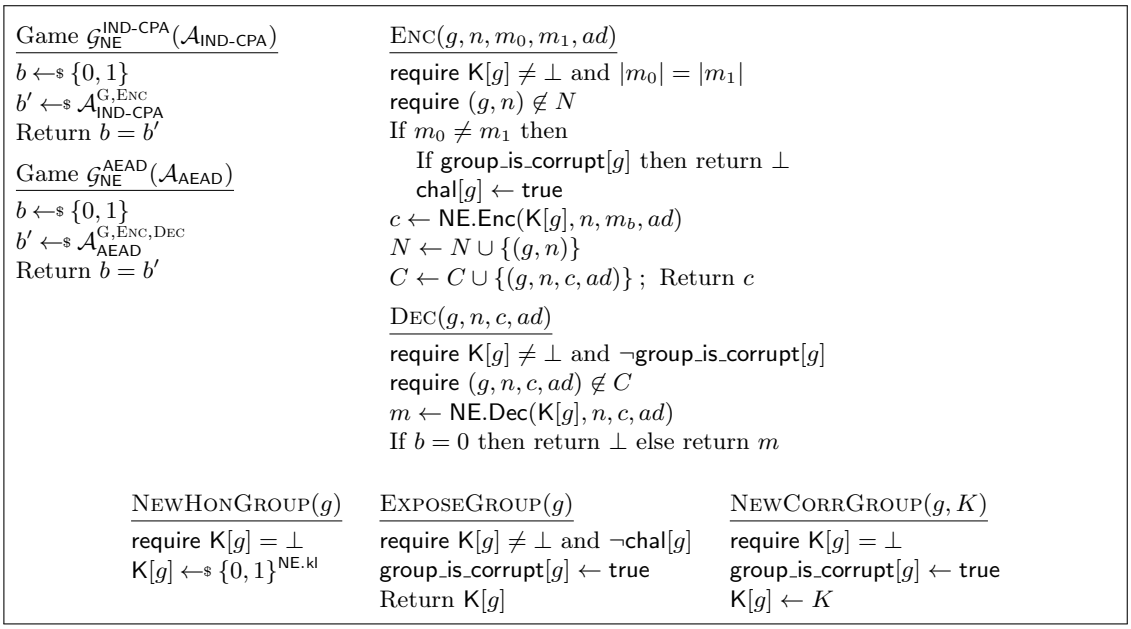


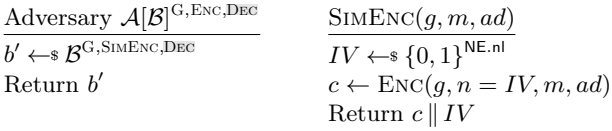
Fig. 1: Games $\mathcal{G}^{\mathsf{IND\text{-}CPA}}$ and $\mathcal{G}^{\mathsf{AEAD}}$ defining the IND-CPA security and the AEAD security of a nonce-based encryption scheme NE. The group oracles $\mathrm{G} = \{\textsc{NewHonGroup}, \textsc{ExposeGroup}, \textsc{NewCorrGroup}\}$ and the encryption oracle Enc are provided to an adversary in both games; the decryption oracle Dec is only provided in the latter.

The game $\mathcal{G}^{\mathsf{IND\text{-}CPA}}$ samples a bit b and requires the adversary to guess it. It is given access to the group oracles G to manage NE keys. The oracle Enc takes a group key identifier$g$, a nonce n, challenge messages m_0 and m_1, and associated data ad and returns the encryption of m_b under the given parameters. To prevent trivial wins, a group cannot be exposed after it has been queried to Enc on $m_0 \neq m_1$, and calls to Enc on exposed groups must satisfy $m_0 = m_1$. We disallow nonce-repetition by requiring that (g, n) is unique across all queries to Enc.

AEAD security of nonce-based encryption schemes. The AEAD security game $\mathcal{G}^{\mathsf{AEAD}}$ for a nonce-based encryption scheme NE and an adversary $\mathcal{A}_{\mathsf{AEAD}}$ is shown in Fig. 1. The advantage of $\mathcal{A}_{\mathsf{AEAD}}$ against the AEAD security of NE is given by $\mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}}) = 2 \cdot \Pr[\mathcal{G}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}})] - 1$. The game $\mathcal{G}^{\mathsf{AEAD}}$ samples a bit b and requires the adversary to guess it. It is given access to the group oracles G to manage NE keys. The encryption oracle Enc is the same as the corresponding oracle in $\mathcal{G}^{\mathsf{IND\text{-}CPA}}$. The decryption oracle Dec takes (g, n, c, ad) as inputs and returns the value of $\mathsf{NE.Dec}(\mathsf{K}[g], n, c, ad)$ if $b = 1$ and $\bot$ otherwise following the style of [RS06, Shr04]. To prevent trivial wins it prevents the adversary from querying corrupt groups or previous outputs of Enc. For encryption schemes with $\mathsf{NE.AD} = \{\varepsilon\}$, this definition captures the authenticated encryption security game $\mathcal{G}^{\mathsf{AE}}$.

Capturing IV-based encryption schemes. We use the same syntax that we defined for nonce-based encryption schemes for random-IV-based encryption schemes — a trick originally pointed out by Rogaway [Rog04]. Then we capture their IND-CPA and AEAD security by restricting to the class of adversaries that we define below.

Let $\mathbb{A}^{\mathsf{NE}}_{\$\text{-}\mathsf{IV}}$ be the class of all adversaries of the form $\mathcal{A}[\mathcal{B}]$ defined as follows.



SUFCMA SECURITY OF DIGITAL SIGNATURE SCHEMES. The strong unforgeability game $\mathcal{G}^{\mathsf{SUFCMA}}$ for a digital signature scheme DS and an adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ is shown in Fig. 2. The advantage of $\mathcal{A}_{\mathsf{SUFCMA}}$ against the SUFCMA security of DS is given by $\mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}}) = \Pr[\mathcal{G}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})]$. Adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ has access to the user oracles to manage signing and verification keys. Additionally, it can query the SIGN oracle which takes a user u and a message m and returns a signature s on m under $\mathsf{sk}[u]$. The adversary is required to produce a signature for an honest u that wasn't previously returned by SIGN.[6]

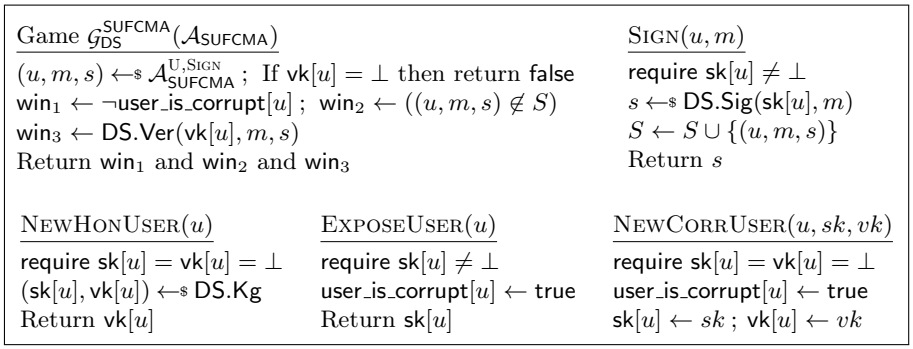


Fig. 2: Game defining strong unforgeability of a digital signature scheme DS, where U = {NEWHONUSER, EXPOSEUSER, NEWCORRUSER}.

3 Symmetric Signcryption

In this section, we recall the syntax and security definitions for symmetric signcryption schemes. We then discuss the secure Sign-then-Encrypt and Encrypt-then-Sign constructions proposed by JKS24. In particular, we enumerate the various types of bindings required by these constructions and describe attacks against different security notions that result from the absence of any of these bindings. In our analysis of different applications throughout this work, we use the secure constructions of JKS24 as a benchmark, identifying attacks that arise from the lack of binding within the underlying symmetric signcryption schemes.

3.1 Symmetric Signcryption Definitions

In symmetric signcryption, a group of users share a symmetric key and each user additionally possesses their own signing key pair. To send a message in the group, a user encrypts the message using the shared symmetric group key and authenticates it using their individual signing key.

SYNTAX. We recall the syntax for a symmetric signcryption scheme SS in Fig. 3. In addition, SS specifies a group key length $\mathsf{SS.gkl} \in \mathbb{N}$, a nonce space $\mathsf{SS.NS}$, and an associated data space $\mathsf{SS.AD}$. The user key generation algorithm $\mathsf{SS.UserKg}$ outputs a pair of keys (sk, vk) where sk is the signing key and vk is the corresponding verification key. The signcryption algorithm $\mathsf{SS.SigEnc}$ takes as input a group key identifier $g \in \{0,1\}^*$, the corresponding symmetric group key $K_g \in \{0,1\}^{\mathsf{SS.gkl}}$, the sender identifier $u \in \{0,1\}^*$, the sender's signing key sk_u, a nonce $n \in \mathsf{SS.NS}$, a message $m \in \{0,1\}^*$, and associated data $ad \in \mathsf{SS.AD}$, and outputs a ciphertext c.[7] The deterministic unsigncryption algorithm $\mathsf{SS.VerDec}$ takes the tuple $(g, K_g, u, vk_u, n, c, ad)$ as input where vk_u is the sender's verification key, and outputs a message $m \in \{0,1\}^* \cup \{\bot\}$ where $\bot$ indicates failure to recover the message.

[6] In the existential version $\mathcal{G}^{\mathsf{EUFCMA}}$ of the game, the adversary must produce a signature for an honest user u and a message m that wasn't queried to SIGN.

[7] JKS24 called g and u the group and user identifier. Our tweaked terminology tries to highlight what they correspond to in the protocols themselves.

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

Fig. 3: Syntax of a symmetric signcryption scheme SS.

The correctness of a symmetric signcryption scheme guarantees that

$$\mathsf{SS.VerDec}(g, K_g, u, vk_u, n, \mathsf{SS.SigEnc}(g, K_g, u, sk_u, n, m, ad), ad) = m$$

for all $g \in \{0,1\}^*$, $K_g \in \{0,1\}^{\mathsf{SS.gkl}}$, $u \in \{0,1\}^*$, $(sk_u, vk_u) \in [\mathsf{SS.UserKg}]$, $n \in \mathsf{SS.NS}$, $m \in \{0,1\}^*$, and $ad \in \mathsf{SS.AD}$.

IN-GROUP UNFORGEABILITY. IUF security of a symmetric signcryption scheme guarantees the authenticity of ciphertexts in the SUFCMA sense. It ensures that an attacker cannot forge a ciphertext that appears to have come from any group member other than themselves. The game is parameterized by a ciphertext-triviality predicate that defines what constitutes a trivial win for the attacker. By using different triviality predicates, one captures the exact authenticity guarantees provided by a given scheme.

We define for the IUF security game $\mathcal{G}^{\mathsf{IUF}}$ for a symmetric signcryption scheme SS, ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}$, and adversary $\mathcal{A}_{\mathsf{IUF}}$ in Fig. 4. The advantage of $\mathcal{A}_{\mathsf{IUF}}$ against the IUF security of SS is given by $\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{SS},\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) = \Pr[\mathcal{G}^{\mathsf{IUF}}_{\mathsf{SS},\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}})]$. The adversary is given access to the user oracles U, the group oracles G, the encryption oracle SIGENC, the decryption oracle VERDEC, and is required to forge a ciphertext for an honest user.

The game uses the user_is_corrupt table to keep track of exposed/corrupt users. Whenever the adversary exposes a user or creates a corrupt one, the game sets user_is_corrupt$[u]$ to true. Analogously, the game updates the group_is_corrupt table whenever the adversary queries its EXPOSEGROUP or NEWCORRGROUP oracle. The IUF game does not use the table thereafter, allowing the adversary to learn and control symmetric group keys.

The SIGENC oracle accepts a tuple (g, u, n, m, ad) as input and returns a ciphertext c generated by running $\mathsf{SS.SigEnc}(g, \mathsf{K}[g], u, \mathsf{sk}[u], n, m, ad)$. It subsequently logs $((g, u, n, m, ad), c)$ into the table C to record all queries made to SIGENC along with the corresponding outputs. The VERDEC oracle takes a tuple (g, u, n, c, ad) and executes $\mathsf{SS.VerDec}(g, \mathsf{K}[g], u, \mathsf{sk}[u], n, c, ad)$ to obtain the message m. If $m \neq \perp$, the VERDEC oracle runs the triviality predicate $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}$ on the set C and the value $z = ((g, u, n, m, ad), c)$ to verify whether the current query constitutes a valid forgery, in which case it sets the win flag.

Ciphertext-triviality predicates are used to parameterize both the IUF and OAE security games. We present definitions for several ciphertext-triviality predicates in Fig. 5, with the strongest of these being $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$; under $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$, any ciphertext that wasn't previously output by SIGENC constitutes a successful forgery. The $\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}$ predicate can be thought of as the existential counterpart to $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$. We also use the predicates $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}group}}_{\mathsf{trivial}}$ and $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ that capture the authenticity of schemes where a ciphertext for (g, u, n, m, ad) does not bind to g and u respectively.

OUT-GROUP AUTHENTICATED ENCRYPTION. The OAE security of a symmetric signcryption scheme guarantees the confidentiality and integrity of ciphertexts in the AEAD sense. It ensures that a group-outsider cannot learn anything about messages sent on the group, nor can it modify the ciphertexts sent by group members. The security definition is parameterized with a ciphertext-triviality predicate to capture the exact guarantees provided by a scheme.

We provide the define for the OAE security game $\mathcal{G}^{\mathsf{OAE}}$ for a symmetric signcryption scheme SS, ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}}$ and adversary $\mathcal{A}_{\mathsf{OAE}}$ in Fig. 4. The advantage of $\mathcal{A}_{\mathsf{OAE}}$ against the OAE security of SS is given by $\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}}) = 2 \cdot \Pr[G^{\mathsf{OAE}}_{\mathsf{SS},\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}})] - 1$. The adversary is given access to the user oracles U, the group oracles G, the encryption oracle SIGENC, and the decryption oracle VERDEC, and is required to guess the challenge bit b. The user and group oracles are the same as the corresponding oracles in the IUF security game, except the EXPOSEGROUP oracle cannot be called

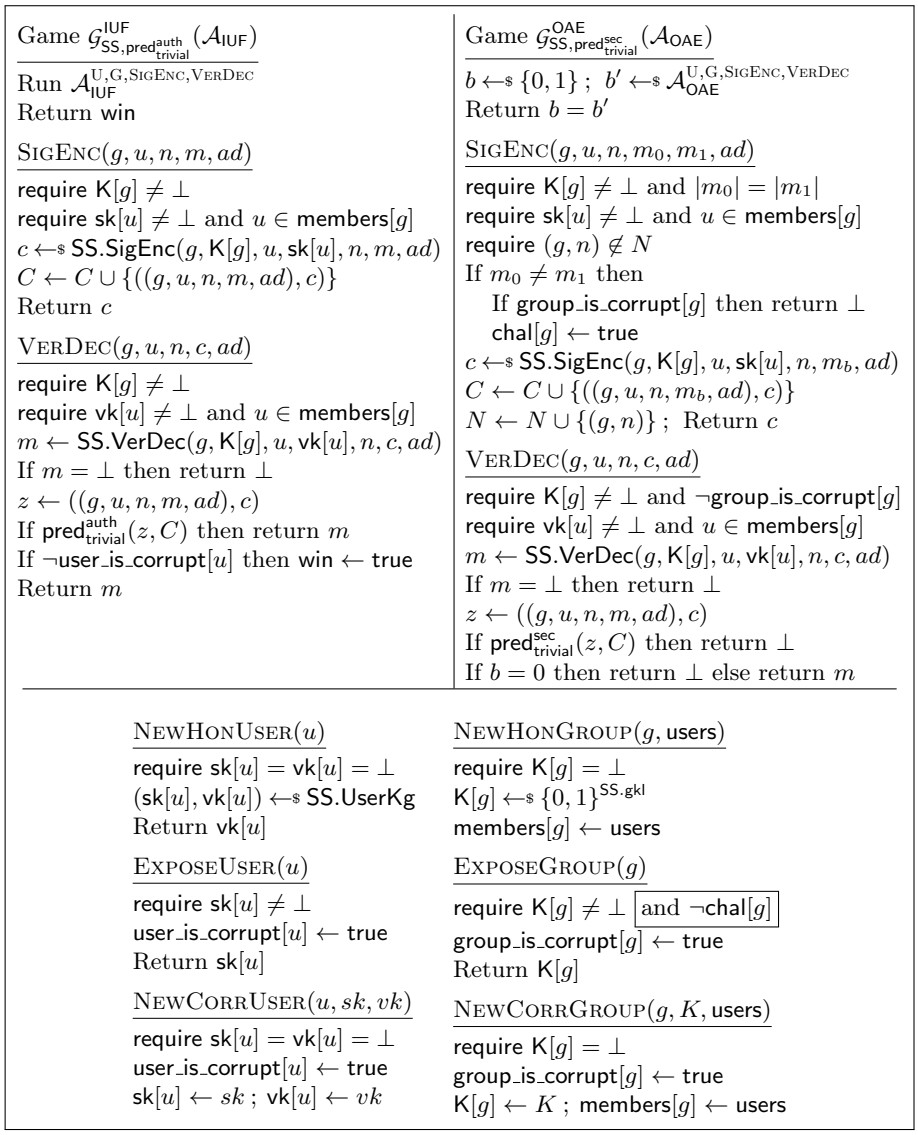


Fig. 4: **Left pane:** Game defining the IUF security of a symmetric signcryption scheme SS with respect to a ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{auth}}_{\mathsf{trivial}}$. **Right pane:** Game defining the OAE security of SS with respect to a ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}}$. **Bottom pane:** User oracles $\mathrm{U} = \{\textsc{NewHonUser}, \textsc{ExposeUser}, \textsc{NewCorrUser}\}$ and group oracles $\mathrm{G} = \{\textsc{NewHonGroup}, \textsc{ExposeGroup}, \textsc{NewCorrGroup}\}$ provided to an adversary in either game, except that the boxed code only appears in the OAE security game.

on a group that was used for a challenge query. The game does not use the user_is_corrupt table, allowing the adversary to corrupt individual users.

The SigEnc oracle takes as input (g,u,n,m_0,m_1,ad) and returns the ciphertext c produced by $\mathsf{SS.SigEnc}(g,\mathsf{K}[g],u,\mathsf{sk}[u],n,m_b,ad)$. As is standard, the oracle requires that the sizes of m_0 and m_1 are equal, i.e., $|m_0| = |m_1|$, and challenge queries must be made on honest groups with the condition that $m_0 \neq m_1$. The oracle uses the table C to record all queries and their outputs. The VerDec oracle takes (g,u,n,c,ad) as input and returns the message m obtained by running $\mathsf{SS.VerDec}(g,\mathsf{K}[g],u,\mathsf{sk}[u],n,c,ad)$. If $m = \bot$, the oracle returns $\bot$. Otherwise, the oracle applies the triviality predicate $\mathsf{pred}^{\mathsf{sec}}_{\mathsf{trivial}}$ to the set C and the tuple $z = ((g,u,n,m,ad),c)$ to check whether the current query is a trivial forgery, in which

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

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

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

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

Fig. 5: Ciphertext-triviality predicates that capture rules for whether a successfully decrypted VerDec query was trivially forgeable.

StE.UserKg	EtS.UserKg
$(sk, vk) \leftarrow\!\$\ \mathsf{DS.Kg}$; Return (sk, vk)	$(sk, vk) \leftarrow\!\$\ \mathsf{DS.Kg}$; Return (sk, vk)
$\underline{\mathsf{StE.SigEnc}(g, K_g, u, sk_u, n, m, ad)}$	$\underline{\mathsf{EtS.SigEnc}(g, K_g, u, sk_u, n, m, ad)}$
$m_s \leftarrow \langle \underline{g}, \underline{n}, m, \underline{ad} \rangle$	$ad_{\mathsf{NE}} \leftarrow \langle \underline{u}, ad \rangle$
$s \leftarrow\!\$\ \mathsf{DS.Sig}(sk_u, m_s)$	$c_{\mathsf{NE}} \leftarrow \mathsf{NE.Enc}(K_g, n, m, ad_{\mathsf{NE}})$
$m_e \leftarrow s \,\|\, m$; $ad_{\mathsf{NE}} \leftarrow \langle \underline{u}, ad \rangle$	$m_s \leftarrow \langle \underline{g}, \underline{n}, c_{\mathsf{NE}}, \underline{ad} \rangle$
$c \leftarrow \mathsf{NE.Enc}(K_g, n, m_e, ad_{\mathsf{NE}})$	$s \leftarrow\!\$\ \mathsf{DS.Sig}(sk_u, m_s)$
Return c	Return (c_{NE}, s)
$\underline{\mathsf{StE.VerDec}(g, K_g, u, vk_u, n, c, ad)}$	$\underline{\mathsf{EtS.VerDec}(g, K_g, u, vk_u, n, c, ad)}$
$ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$	$(c_{\mathsf{NE}}, s) \leftarrow c$
$m_e \leftarrow \mathsf{NE.Dec}(K_g, n, c, ad_{\mathsf{NE}})$	$m_s \leftarrow \langle g, n, c_{\mathsf{NE}}, ad \rangle$
If $m_e = \perp$ then return $\perp$	If $\neg\mathsf{DS.Ver}(vk_u, m_s, s)$ then return $\perp$
$s \,\|\, m \leftarrow m_e$ // s.t. $\|s\| = \mathsf{DS.sl}$, $\|m\| \geq 0$	$ad_{\mathsf{NE}} \leftarrow \langle u, ad \rangle$
$m_s \leftarrow \langle g, n, m, ad \rangle$	$m \leftarrow \mathsf{NE.Dec}(K_g, n, c_{\mathsf{NE}}, ad_{\mathsf{NE}})$
If $\neg\mathsf{DS.Ver}(vk_u, m_s, s)$ then return $\perp$	Return m
Return m	

Fig. 6: **Left pane:** Signcryption scheme $\mathsf{StE} = \mathsf{SIGN\text{-}THEN\text{-}ENCRYPT\text{-}SS}[\mathsf{DS}, \mathsf{NE}]$. **Right pane:** Symmetric signcryption scheme $\mathsf{EtS} = \mathsf{ENCRYPT\text{-}THEN\text{-}SIGN\text{-}SS}[\mathsf{NE}, \mathsf{DS}]$. We use underlined blue font to highlight the explicit binding of components.

case it immediately returns $\perp$. The final line of the VerDec oracle is executed only if $m \neq \perp$ and the query was not deemed a trivial forgery. In that case, the oracle returns $\perp$ if $b = 0$ and m otherwise.

We note the modifications we made to the OAE security definition of JKS24. We removed the checks for nonce-misuse resistance, as the schemes we analyze are not misuse-resistant. In our definition, the VerDec oracle returns $\perp$ in both cases: when it encounters a trivial forgery and when decryption fails. JKS24 parameterized their OAE security game with an *output-guarding function* such that the VerDec oracle would return the output of this function for trivial forgeries and $\perp$ for decryption failures. The strongest output guarding function allows the adversary to distinguish between triggering the triviality predicate and causing a decryption failure, while the weakest function always returns $\perp$. We capture the OAE security of a random-IV-based symmetric signcryption schemes by restricting to a class $\mathbb{A}^{\mathsf{SS}}_{\$\text{-}\mathsf{IV}}$ of adversaries defined analogously to $\mathbb{A}^{\mathsf{NE}}_{\$\text{-}\mathsf{IV}}$ in Section 2.

3.2 Secure Constructions and the Importance of Binding

We recall the secure constructions, StE and EtS, proposed by JKS24 in Fig. 6. The StE scheme is a Sign-then-Encrypt construction that uses a digital signature scheme DS and a nonce-based encryption scheme NE as building blocks. The IUF security of StE with respect to the predicate $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ reduces to the SUFCMA security of DS and the OAE security of StE with respect to $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ reduces to the AEAD security of NE. The EtS scheme is an Encrypt-then-Sign construction built from the same building blocks as StE. The IUF security of EtS with respect to $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ similarly reduces to the SUFCMA security of DS

Attack-1 $\mathcal{A}_1^{\mathrm{U,G,SigEnc,VerDec}}$	Attack-2 $\mathcal{A}_2^{\mathrm{U,G,SigEnc,VerDec}}$
$\textsc{NewCorrGroup}(g_0, K, \mathsf{users})$	$\textsc{NewCorrGroup}(g, K, \mathsf{users})$
$\textsc{NewCorrGroup}(g_1, K, \mathsf{users})$	$n_0 \leftarrow 0^{\mathsf{NE.nl}}$; $n_1 = 1^{\mathsf{NE.nl}}$
$n \leftarrow 0^{\mathsf{NE.nl}}$; $ad \leftarrow \varepsilon$	$ad \leftarrow \varepsilon$
$c \leftarrow \textsc{SigEnc}(g_0, u_i, n, 0, ad)$	$c_0 \leftarrow \textsc{SigEnc}(g, u_i, n_0, 0, ad)$
$\textsc{VerDec}(g_1, u_i, n, c, ad)$	$s \,\|\| m \leftarrow \mathsf{NE.Dec}(K, n_0, c_0, \langle u_i, ad\rangle)$
	$c_1 \leftarrow \mathsf{NE.Enc}(K, n_1, s \,\|\| m, \langle u_i, ad\rangle)$
	$\textsc{VerDec}(g, u_i, n_1, c_1, ad)$

Fig. 7: **Left pane:** Attack-1 against the IUF security of StE-except-group with respect to $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$. **Right pane:** Attack-2 against the IUF security of StE-except-nonce with respect to $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$. For each of the attacks, we fix $n \geq 2, i \in [0, n)$, a list of sender identifiers $\mathsf{users} = [u_0, u_1, ..., u_n]$, and a message m.

and the OAE security of EtS with respect to the weaker $\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}$ predicate reduces to the AEAD security of NE. The OAE security of EtS with respect to $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ would need to assume that DS produces unique signatures.

The importance of binding. Binding refers to the inclusion of the symmetric encryption parameters into the signature input, and vice versa, the inclusion of the digital signature parameters into the encryption input. We highlight the two types of binding in the StE and EtS using underlined blue font in Fig. 6.

In both constructions, the input m_s to the signature scheme includes the group key identifier g, the nonce n, and the associated data ad. Similarly, the associated data ad_{NE} input to the encryption scheme contains the sender identifier u. This binds the identity of a specific group instance to the signature and the identity of the sender to the symmetric ciphertext. Achieving strong IUF and OAE security in a symmetric signcryption scheme requires proper binding between the underlying components. To highlight this, we show attacks against the IUF and OAE security of a symmetric signcryption scheme with respect to the $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ predicate, which would arise from the absence of proper binding.

Consider the symmetric signcryption scheme StE-except-group which is identical to the StE construction except m_s does not include g. In Fig. 7, we describe an attack Attack-1 in which adversary $\mathcal{A}_1$ breaks the IUF security of StE-except-group. It begins by creating two corrupt groups, g_0 and g_1, with the same symmetric key K. Next, it queries its SigEnc oracle on (g_0, u, n, m, ad), and obtains the ciphertext c. It queries the VerDec oracle on (g_1, u, n, c, ad). The ciphertext c would decrypt and verify successfully since $K_{g_0} = K_{g_1}$ and the remaining arguments are the same as the SigEnc query. Furthermore, the $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ predicate evaluates to false since $((g_1, u, n, m, ad), c) \notin C$. Thus, the adversary wins the IUF security game achieving $\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{StE\text{-}except\text{-}group}, \mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_1) = 1$.

Next we consider the schemes StE-except-nonce and StE-except-user that are identical to StE except m_s does not include n in the former and ad_{NE} does not include u in the latter. We show Attack-2 and Attack-3 against the IUF and OAE security of these schemes in Figs. 7 and 8 respectively such that $\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{StE\text{-}except\text{-}nonce}, \mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_2) = 1$ and $\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{StE\text{-}except\text{-}user}, \mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_3) = 1$. While our examples focus on versions of the StE construction, similar attacks apply to variants of the EtS construction that lack proper binding.

The takeaway is that proper binding is necessary for achieving strong IUF and OAE security. This requirement becomes apparent during analysis. For each construction, we examine whether the signature properly binds to g, n, and ad, and whether the symmetric ciphertext binds to u. Whenever a construction fails to explicitly bind these components, we prove security against weak predicates of the form $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}}}_{\mathsf{trivial}}$/$\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}}}_{\mathsf{trivial}}$ which reflect security up to the corresponding attack.

The lack of explicit binding doesn't always lead to a practical exploit. Determining whether a theoretical attack is exploitable requires careful examination of how the construction is used in the broader protocol. This involves identifying whether the protocol imposes implicit bindings that can used to provably achieve strong security when analyzing the construction's composition with other primitives.

Attack-3 $\mathcal{A}_3^{\mathrm{U},\mathrm{G},\mathrm{SigEnc},\mathrm{VerDec}}$

$\mathrm{NewCorrUser}(u_0, sk, vk)$
$\mathrm{NewCorrUser}(u_1, sk, vk)$
$\mathrm{NewHonGroup}(g, \{u_0, u_1\})$
$n \leftarrow 0^{\mathsf{NE.nl}}$
$c \leftarrow \mathrm{SigEnc}(g, u_0, n, 0, 1, \varepsilon)$
$m \leftarrow \mathrm{VerDec}(g, u_1, n, c, \varepsilon)$
If $m \neq \perp$ then return 1
Else return 0

Fig. 8: Attack-3 against the OAE security of StE-except-user with respect to $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$. We fix identifiers $u_0 \neq u_1$, and g.

JKS24 observed implicit binding during their analysis of Keybase's header encryption algorithm, SealPacket, that did not explicitly bind to the user or group key identifier. But within the BoxMessage chat encryption algorithm, it was always invoked on messages containing these identifiers, thereby providing implicit binding. This allowed JKS24 to prove that BoxMessage achieves strong IUF and OAE security even though SealPacket in isolation does not.

4 Analysis of Sign-then-Encrypt Constructions

4.1 MLS

An overview of MLS. In this section we present our analysis of the symmetric signcryption scheme underlying MLS. We begin by providing a high level overview of MLS. MLS maintains three main data structures to track the membership states in a dynamic group and derive corresponding shared group secrets such that each new message sent in the group is encrypted with a fresh symmetric key. These are the ratchet tree, the key schedule, and the secret tree. We provide a summary of these data structures and their interactions in Fig. 9.

Consider a messaging group G consisting of $|G|$ group members $u_0, ..., u_{|G|-1}$ during an epoch ep. An epoch represents a particular membership state. A new epoch is initiated whenever a member is added or removed from the group or when they updates their individual cryptographic state. The *ratchet* tree for a group describes the membership of the group during a given epoch. It is a full binary tree that is built deterministically in a bottom-up manner.[8] Leaf nodes in the ratchet tree are assigned an index $\mathsf{leafIndex} \in \mathbb{Z}_{|G|}$, starting at 0 from the left to $|G| - 1$ at the right, corresponding to the members $u_0, u_1, ..., u_{|G-1|}$ respectively. Each $\mathsf{leafIndex}$ corresponds to exactly one user. The root of the ratchet tree is used to derive the commit secret k_{commit} that seeds the key schedule.

The *key schedule* maintains the shared symmetric group secret k_{epoch}, called the epoch secret, for every epoch. It propagates the epoch secret from epoch to epoch using the epoch secret from the previous epoch and the commit secret for the current epoch. Each member uses the epoch secret to derive the encryption secret k_{es} that forms the root of the secret tree.

Group members use the *secret tree* to derive per-message symmetric keys and nonces. Each member of the group uses the encryption secret k_{es} as the root and deterministically derives the secret tree in a top-down manner.[9] Each node in the secret tree contains a symmetric key and each leaf in the secret tree represents the same group member as the corresponding leaf in the ratchet tree. The symmetric key k_u at a leaf corresponding to a user u seeds two symmetric key chains (ratchets) namely the handshake ratchet HR and the application ratchet AR. These ratchets in turn derive per-message keys and nonces for handshake and application messages. We only consider application messages but our results should extend to handshake messages.

[8] Details about the ratchet tree derivation are provided in Section 7 of [BBR+23].
[9] Details about the secret tree derivation are provided in Section 9 of [BBR+23].

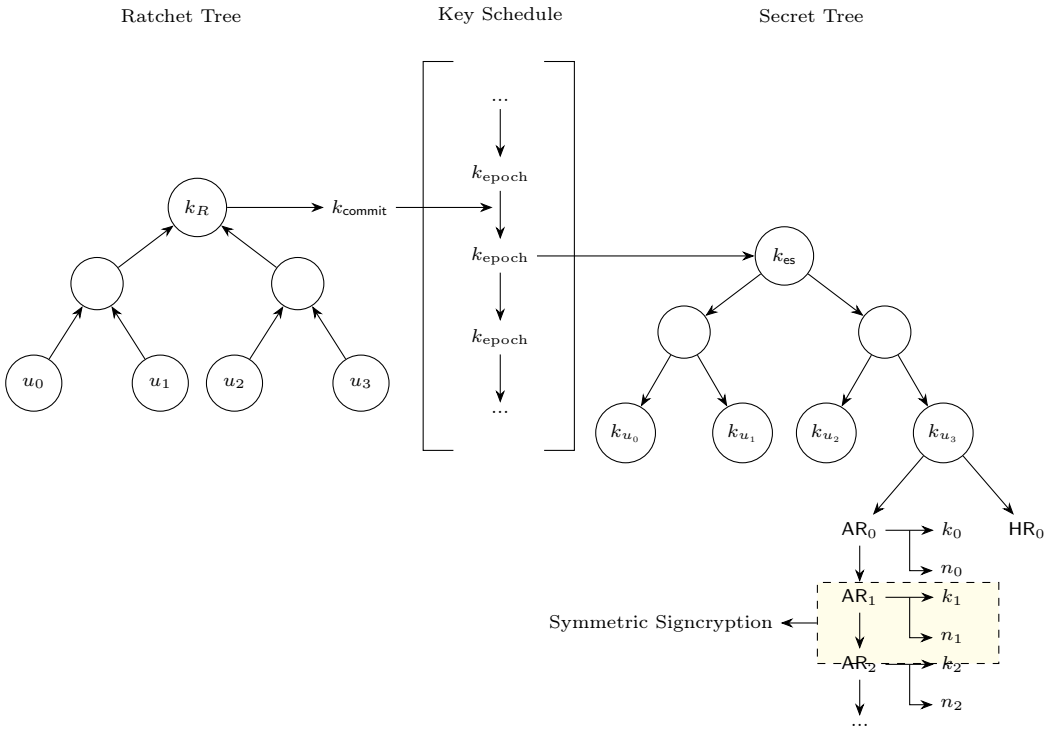


Fig. 9: The ratchet tree, key schedule, and secret tree in MLS. Here k_{R} is the root of the ratchet tree, k_{commit} is derived from k_{R} and propagates the key schedule, and k_{es} is the encryption secret for a group G and epoch ep where u_0, u_1, u_2, and u_3 are group members. HR and AR are the handshake and application ratchets.

A sender $u \in G$ uses the ith key-nonce pair (k_i, n_i) in their application ratchet to AEAD encrypt the ith message they send to the group during a given epoch ep. They then advance the generation i for the next message, ensuring that the pair (k_i, n_i) is only used once. The user u, the key k_i and the nonce n_i can be uniquely identified by the tuple $(G, \mathsf{leafIndex}_u, \mathsf{ep}, i)$.

Chat Encryption in MLS. Chat encryption in MLS composes a digital signature scheme and a nonce-based encryption scheme in the Sign-then-Encrypt fashion. We show pseudocode for the corresponding subroutine, which we call $\mathsf{MLSSendPrivateMessage}$, in Fig. 10.

Consider a sender u with leaf index $\mathsf{leafIndex}_u$ in a group G during an epoch ep that wants to send their i^{th} message m and associated data ad to the group. The $\mathsf{MLSSendPrivateMessage}$ subroutine works as follows. First it constructs the authentication message m_{DS} by uniquely encoding $G, e, \mathsf{leafIndex}_u, ad$, and m. It then signs m_{DS} using the sender's signing key sk_u to produce the signature s.

Next, it builds the input m_{AEAD} for AEAD encryption as $m \,\|\, s$ and the associated data ad_{AEAD} as the unique encoding of G, ep, and ad. It uses the current epoch secret k_{ep} to derive the encryption secret k_{es} using a key derivation function KDF. Using k_{es} it derives the secret tree for the current epoch and retrieves the symmetric key k_u corresponding to the sender u from the leaf node at $\mathsf{leafIndex}_u$. It then builds application ratchet secrets using KDF starting with k_u until AR_i corresponding to the current generation i. It uses AR_i and KDF to derive the one-time encryption key-nonce pair (k_{AEAD}, n). To protect against nonce reuse in case a client accidentally repeats a value of i, it masks the nonce n using a random 32-bit "reuse guard" r, producing the final nonce n_{AEAD}. Finally, it encrypts m_{AEAD} and ad_{AEAD} using an AEAD scheme with k_{AEAD} and n_{AEAD} to obtain the ciphertext c_{AEAD}, which is broadcast to the group.

Symmetric Signcryption in MLS. Every message sent on a group can be uniquely identified by the tuple $(G, \mathsf{ep}, \mathsf{leafIndex}_u, i)$. We use this to capture each of the aforementioned single-use symmetric sign-

MLSSendPrivateMessage

// Authentication

$m_{\mathsf{DS}} \leftarrow \langle G, \mathsf{ep}, \mathsf{leafIndex}, ad, m\rangle$; $s \leftarrow\!\!\$\ \mathsf{DS.Sig}(sk_u, m_{\mathsf{DS}})$

// Encryption

$m_{\mathsf{AEAD}} \leftarrow m \,\|\, s$; $ad_{\mathsf{AEAD}} \leftarrow \langle G, \mathsf{ep}, ad\rangle$; $j \leftarrow 0$

$k_{\mathsf{es}} \leftarrow \mathsf{KDF.Exp}(k_{\mathsf{ep}}, (\mathsf{KDF.Ext.ol}, \text{“MLS1.0”} \,\|\, \text{“encryption”}), \mathsf{KDF.Ext.ol})$

[Generate secret tree from k_{es} and go to $\mathsf{leafIndex}_u$ to obtain k_u]

$\mathsf{AR}_0 \leftarrow \mathsf{KDF.Exp}(k_u, (\mathsf{KDF.Ext.ol}, \text{“MLS1.0”} \,\|\, \text{“secret”}, j), \mathsf{KDF.Ext.ol})$

while $j < i$ do

  $j \leftarrow j + 1$

  $\mathsf{AR}_j \leftarrow \mathsf{KDF.Exp}(\mathsf{AR}_{j-1}, (\mathsf{KDF.Ext.ol}, \text{“MLS1.0”} \,\|\, \text{“secret”}, j), \mathsf{KDF.Ext.ol})$

$k_{\mathsf{AEAD}} \leftarrow \mathsf{KDF.Exp}(\mathsf{AR}_i, (\mathsf{AEAD.kl}, \text{“MLS1.0”} \,\|\, \text{“key”}, i), \mathsf{AEAD.kl})$

$n \leftarrow \mathsf{KDF.Exp}(\mathsf{AR}_i, (\mathsf{AEAD.nl}, \text{“MLS1.0”} \,\|\, \text{“nonce”}, i), \mathsf{AEAD.nl})$

$r \leftarrow\!\!\$\ \{0,1\}^{32}$; $n_{\mathsf{AEAD}} \leftarrow n \oplus r$

$c_{\mathsf{AEAD}} \leftarrow \mathsf{AEAD.Enc}(k_{\mathsf{AEAD}}, n_{\mathsf{AEAD}}, m_{\mathsf{AEAD}}, ad_{\mathsf{AEAD}})$

Fig. 10: The MLSSendPrivateMessage subroutine. G is the group name, u is the sender, sk_u is the sender's signing key, ep is the epoch, k_{ep} is the shared group secret, i is the message generation, m is the message, and ad is the associated data. Pink highlighting marks lines that are later modeled as a symmetric signcryption scheme. [Skipped lines] refer to the derivation of the secret tree in Fig. 9.

MLS-StE.UserKg

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

MLS-StE.SigEnc$(g, K_g, u, sk_u, n, m, ad)$

$(g.\mathsf{Id}, g.\mathsf{ep}, g.\mathsf{leafIndex}, g.\mathsf{generation}) \leftarrow g$

$m_{\mathsf{DS}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, g.\mathsf{leafIndex}, ad, m\rangle$

$s \leftarrow\!\!\$\ \mathsf{DS.Sig}(sk_u, m_{\mathsf{DS}})$

$m_{\mathsf{NE}} \leftarrow m \,\|\, s$

$ad_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, ad\rangle$

$c \leftarrow \mathsf{NE.Enc}(K_g, n, m_{\mathsf{NE}}, ad_{\mathsf{NE}})$

Return c

MLS-StE.VerDec$(g, K_g, u, vk_u, n, c, ad)$

$ad_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, ad\rangle$

$m_{\mathsf{NE}} \leftarrow \mathsf{NE.Dec}(K_g, n, c, ad_{\mathsf{NE}})$

If $m_{\mathsf{NE}} = \perp$ then return $\perp$

$m \,\|\, s \leftarrow m_{\mathsf{NE}}$ // s.t. $|s| = \mathsf{DS.sl}$, $|m| \geq 0$

$m_{\mathsf{DS}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, g.\mathsf{leafIndex}, ad, m\rangle$

If $\neg\mathsf{DS.Ver}(vk_u, m_{\mathsf{DS}}, s)$ then return $\perp$

Return m

Fig. 11: Scheme MLS-StE = MLS-SIGN-THEN-ENCRYPT[DS, NE] modelling symmetric signcryption in MLS.

cryption schemes by defining a *symmetric signcryption group key identifier* object. For MLS, we define a symmetric signcryption group key identifier g as $g = (g.\mathsf{Id}, g.\mathsf{ep}, g.\mathsf{leafIndex}, g.\mathsf{generation})$ where $g.\mathsf{Id} = G, g.\mathsf{ep} = \mathsf{ep}, g.\mathsf{leafIndex} = \mathsf{leafIndex}_u$, and $g.\mathsf{generation} = i$. A messaging group G at the application layer will have multiple corresponding symmetric signcryption groups g in this model. The membership of the symmetric signcryption group g is the same as that of the messaging group G during epoch ep. For each symmetric signcryption group g, group members share a symmetric encryption key K_g and a nonce n that correspond to k_{AEAD} and n_{AEAD} respectively in Fig. 10. Our construction MLS-StE uses symmetric signcryption groups to model the highlighted parts of MLSSendPrivateMessage as a symmetric signcryption scheme as follows.

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

Security Analysis of MLS-StE We identify three attacks, two against the IUF and one against the OAE security of MLS-StE, that result from insufficient binding DS and NE. One attack against the IUF security

is practically exploitable, but the composition of MLS-StE with higher level primitives mitigates the other two. We show that MLS-StE achieves weaker versions of IUF and OAE security.

Attacks against IUF and OAE security of MLS-StE. Consider the IUF security of MLS-StE for a symmetric signcryption group $g = (G, \mathsf{ep}, \mathsf{leafIndex}, \mathsf{generation})$ and a user u. An MLS-StE ciphertext c encrypts $m \,\|\, s$ using the key K_g and the nonce n where s is a signature over $\langle G, \mathsf{ep}, \mathsf{leafIndex}, ad, m\rangle$. Notably, s does not authenticate the message generation $\mathsf{generation}$[10] or the nonce n used for encryption. For optimal IUF security, the signature should bind to the symmetric signcryption group key identifier g, the nonce n, and the associated data ad. In MLS-StE, s does not fully bind to g or n. Consequently, MLS-StE becomes vulnerable to variants of Attack-1 and Attack-2 described in Section 3.2. For the remainder of this section, we refer to these attacks as MLS-Attack-1 and MLS-Attack-2 respectively.

Now consider the OAE security of MLS-StE for a symmetric signcryption group g and a user u. The MLS-StE.SigEnc algorithm never uses its input u. So the ciphertext c does not explicitly bind to the identifier of the sender. As a result, MLS-StE is potentially vulnerable to a variant of Attack-3 discussed in Section 3.2. We refer to this attack as MLS-Attack-3.

Practical attacks and mitigations. We discuss which of the theoretical attacks lead to real-world exploits and which are mitigated by the broader protocol.

MLS-Attack-1 **translates into insider replay and reordering attacks against MLS.** MLS-Attack-1 exploits the lack of binding between the signature s and the group identifier g. Let u and u' be members of an MLS messaging G during an epoch ep. In the IUF security game, the adversary knows the symmetric group key K_g. In MLS, this translates to every group member knowing the application ratchet secrets (and hence the encryption keys) for all members. A malicious member u' can pre-compute the encryption key K_{g_f} and nonce n_f for a symmetric signcryption group $g_f = (G, \mathsf{ep}, \mathsf{leafIndex}_u, i_f)$ where i_f is a future message generation. The member u' could then replay $m \,\|\, s$ produced by u during a previous message generation $i_p < i_f$ by re-encrypting them using K_{g_f} and n_f, impersonating u to the group G. This attack allows a malicious member to replay messages and from other members to the group within an epoch. The attacker can also use the same decrypt-then-reencrypt technique to reorder messages by encrypting $m \,\|\, s$ with key K_{g_p} and nonce n_p for a symmetric signcryption group $g_p = (G, \mathsf{ep}, \mathsf{leafIndex}_u, i_p)$ where i_p is a past message generation.

MLS-Attack-2 **allows an insider to modify a ciphertext without modifying the plaintext.** Group members locally derive encryption keys and nonces from the application ratchet, then xor the first 32 bits of the nonce with the reuse guard for decryption. The reuse guard is authenticated by sender data encryption which uses a symmetric key known to all group members. An attacker can re-encrypt $m \,\|\, s$ under a new nonce $n \oplus r'$ getting ciphertext c' and encrypt the new reuse guard r' as part of a new sender data encryption. This gives an "integrity of ciphertext" (but not "integrity of plaintext") style attack, which does not seem to provide any value against MLS, but it could if an application erroneously assumed MLS provided integrity of ciphertexts against group insiders.

Implicit binding between c and u mitigates MLS-Attack-3. This attack results from the lack of explicit binding between c and u. It is not practically exploitable because MLS provides implicit binding. In MLS groups, the mapping between sender identifiers and leaf indices is bijective. Therefore, a symmetric signcryption group $g = (G, \mathsf{ep}, \mathsf{leafIndex}, i)$ uniquely identifies the sender u. A bad implementation may not uniquely map symmetric signcryption groups to sender identifiers, making MLS-Attack-3 exploitable.

Proving weak IUF security of MLS-StE. We relax the IUF security definition by using a weak predicate $\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}$ defined in Fig. 12. Essentially, the predicate disallows the adversary from winning the IUF security game by executing MLS-Attack-1, MLS-Attack-2, or MLS-Attack-3. We provide some intuition for how we arrived at $\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}$. The $\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}$ predicate defined in Section 3 captures the existential IUF security of a scheme. Under $\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}$, an attacker cannot win by producing a new ciphertext for a message that was previously queried to SigEnc, even if it uses a different nonce (this explains the c', and n'). We then further relax $\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}$ to account for the lack of group binding

[10] The generation *is* authenticated as part of the sender data encryption specified in Section 6.3.2 of [BBR+23], but this only uses a symmetric key known to all group members.

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

Fig. 12: The ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}$.

(this explains the g' term). Now, the attacker must forge a signature to win the game which is prevented by the SUFCMA security of DS.

Theorem 1. *Let* $\mathsf{MLS\text{-}StE} = \mathsf{MLS\text{-}SIGN\text{-}THEN\text{-}ENCRYPT}[\mathsf{DS}, \mathsf{NE}]$ *be the symmetric signcryption scheme built from some* NE *and* DS *as specified in Construction 1. Let* $\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 12. Let* $\mathcal{A}_{\mathsf{IUF}}$ *be any adversary against the* IUF *security of* MLS-StE *with respect to* $\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}$*. Then we can build an adversary* $\mathcal{A}_{\mathsf{DS}}$ *such that*

$$\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{MLS\text{-}StE},\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{DS}}).$$

Proving weak OAE security of MLS-StE. We relax the OAE security definition by using the weak predicate $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$.We show that the OAE security of MLS-StE with respect to $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ reduces to the AEAD security of NE.

Theorem 2. *Let* $\mathsf{MLS\text{-}StE} = \mathsf{MLS\text{-}SIGN\text{-}THEN\text{-}ENCRYPT}[\mathsf{DS}, \mathsf{NE}]$ *be the symmetric signcryption scheme built from some* NE *and* DS *as specified in Construction 1. Let* $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 5. Let* $\mathcal{A}_{\mathsf{OAE}}$ *be any adversary against the* OAE *security of* MLS-StE *with respect to* $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ *and* $\mathsf{func}^{\perp}_{\mathsf{out}}$*. Then we can build an adversary* $\mathcal{A}_{\mathsf{AEAD}}$ *such that*

$$\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{MLS\text{-}StE},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}}) \leq \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{NE}}).$$

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

Discussion. Chat encryption in MLS has been previously studied by Alwen, Coretti, Dodis, and Tselekounis [ACDT21] (ACDT), and by Klimovic [Kli21]. Neither identified the replay attack MLS-Attack-1 that we introduced earlier in this section.

ACDT model MLS chat encryption using an Encrypt-then-Sign approach, whereas the protocol specification [BBR+23], uses Sign-then-Encrypt. Furthermore, while the current MLS specification does not sign message generations, the pseudocode of ACDT does. This raises an important question: *If ACDT had followed the exact MLS specification, would their formalism have identified the* MLS-Attack-1 *replay attack?* We argue that it would not. The definitions of ACDT do not account for insider security. Their security game includes a corruption oracle that reveals the entire state of a group member, including their signing key and all symmetric keys for the current epoch. Moreover, their security definition requires the attacker to be passive after *any* compromise until the group secret is updated. In contrast, MLS-Attack-1 exploits the scenario where the attacker actively injects messages in the group after learning the symmetric key (but not the sender's signing key), a situation not addressed by ACDT.

Klimovic modeled chat encryption correctly as a Sign-then-Encrypt construction, but did not provide a formal analysis of this composition or address insider security notions. Instead, their work primarily aimed to simplify the MLS protocol specification and formalize forward-secure AEAD in group settings.

We underscore the importance of insider security particularly in the context of MLS by referencing Section 8.3.1.1 of the MLS Architecture document [BRO+24] that specifically guarantees the following.

> Compromise of the AEAD keys allows the attacker to send an encrypted message using that key, but cannot send a message to a group which appears to be from any valid client since they cannot forge the signature.

EncryptUsingSessionLGProtocol

$m_{\mathsf{Sig}} \leftarrow m \,\|\, vk_u \,\|\, gpk$
$s \leftarrow \mathsf{Ed25519.Sig}(sk_u, m_{\mathsf{Sig}})$
$m_{\mathsf{Enc}} \leftarrow m \,\|\, vk_u \,\|\, s$
$(epk, esk) \leftarrow\!\!\$\ \mathsf{X25519.Kg}$
$n_{\mathsf{Enc}} \leftarrow \mathsf{Blake2b}(epk \,\|\, gpk)$
$s \leftarrow gpk^{esk}$; $k_{\mathsf{Enc}} \leftarrow \mathsf{HSalsa20}(s, 0^{256})$
$c \leftarrow \mathsf{XSalsa20\text{-}Poly1305.Enc}(k_{\mathsf{Enc}}, n_{\mathsf{Enc}}, m_{\mathsf{Enc}})$

Fig. 13: Here, gpk is the group's public key, (sk_u, vk_u) is the sender's signing key pair, m is the message.

Isolating the symmetric signcryption primitive allows for fine grained analysis of security under the compromise of different keys while other keys remain secret, allowing us to capture the insider replay attack against MLS.

Disclosure. On February 3, 2025, we shared our findings with the MLS working group by reaching out to the MLS mailing list (mls@ietf.org). Two of the RFC authors acknowledged our insider replay attack, while other researchers proposed potential updates to the RFC in light of our findings. There was some discussion on whether the MLS protocol should protect against replay attacks. As a result of this discussion, we were invited to present our findings at the IETF 122 meeting, to provide the working group an opportunity to consider replay attacks and associated revisions to the RFC.

4.2 Session

In this section we present our analysis of the symmetric signcryption scheme underlying Session's protocol as deployed at the time of writing, now called Legacy Groups (LG). Group chat encryption in Session has since updated to a new protocol, GroupsV2. We begin by providing a brief overview of chat encryption in Legacy Groups. It uses a public-key encryption scheme built using the KEM/DEM paradigm [CS03] along with digital signatures such that each message from a sender to a group is encrypted with a fresh symmetric key.

In Session, each user u owns a long-term Ed25519 [BDL+12] signing key pair (sk_u, vk_u). When a user creates a group, they generate an X25519 [Ber06] encryption key pair (gpk, gsk) called the *group encryption key pair* and share it with all members of the group over existing 2-party channels.

Chat encryption in Session. Session uses the EncryptUsingSessionLGProtocol subroutine [Sesb] for chat encryption.[11] We show pseudocode for it in Fig. 13. It is a Sign-then-Encrypt construction, and uses the Ed25519 signature scheme, the XSalsa20-Poly1305 [Ber05, Ber08] nonce-based encryption scheme, and the Blake2b [SA15] and HSalsa20 [Ber08] hash functions as building blocks.

It begins by constructing the signature input m_{Sig} as $m \,\|\, vk_u \,\|\, gpk$ of the message, the sender's verification key, and the group's public key. It then calls Ed25519.Sig to sign m_{Sig} using sk_u to obtain the signature s. Then it builds the input m_{Enc} to the encryption scheme as $m \,\|\, vk_u \,\|\, s$. It samples a fresh, ephemeral X25519 key pair (epk, esk). It derives the nonce n_{Enc} by applying the Blake2b hash function to $epk \,\|\, gpk$ and the key k_{Enc} by applying the HSalsa20 hash function to 0^{256} where HSalsa20 is keyed using $s = gpk^{esk}$. Finally, it calls XSalsa20-Poly1305.Enc to encrypt m_{Enc} using k_{Enc} and n_{Enc} then broadcasts the resulting ciphertext c to the group.

Symmetric Signcryption in Session. Each message in Session is encrypted with a fresh symmetric key that is uniquely defined by (gpk, epk). We use this observation to define symmetric signcryption groups for Session as $g = (g.\mathsf{gpk} = gpk, g.\mathsf{epk} = epk)$ and assume that the associated symmetric key K_g is independently sampled and between all group members. We define the symmetric signcryption scheme SesLG-StE that models the EncryptUsingSessionLGProtocol subroutine as follows.

[11] Session provides a white paper [JSH24] for its protocol, but notes that the groups protocol defined in the paper is not the one used in the live implementation.

SesLG-StE.UserKg

$(sk, vk) \leftarrow\!\!\$\ \mathsf{DS.Kg}$; Return $((sk, vk), \varepsilon)$

SesLG-StE.SigEnc$(g, K_g, u, sk, n, m, ad)$

// $K_g \in \{0,1\}^{256}$; $n \in \{0,1\}^{192}$; $ad = \varepsilon$
$(g.\mathsf{gpk}, g.\mathsf{epk}) \leftarrow g$; $(sk_u, vk_u) \leftarrow sk$
$m_{\mathsf{DS}} \leftarrow m \,\|\, vk_u \,\|\, g.\mathsf{gpk}$
$s \leftarrow \mathsf{DS.Sig}(sk_u, m_s)$
// DS = Ed25519
$m_{\mathsf{NE}} \leftarrow m \,\|\, vk_u \,\|\, s$
$c \leftarrow \mathsf{NE.Enc}(K_g, n, m_{\mathsf{NE}})$
// NE = XSalsa20-Poly1305
Return c

SesLG-StE.VerDec$(g, K_g, u, vk_u, n, c, ad)$

$(g.\mathsf{gpk}, g.\mathsf{epk}) \leftarrow g$
$m_{\mathsf{NE}} \leftarrow \mathsf{NE.Dec}(K_g, n, c)$
If $m_{\mathsf{NE}} = \perp$ then return $\perp$
$m \,\|\, vk_u \,\|\, s \leftarrow m_{\mathsf{NE}}$
// s.t. $|s| = \mathsf{DS.sl}$, $|vk_u| = \mathsf{DS.vkl}$, $|m| \geq 0$
$m_{\mathsf{DS}} \leftarrow m \,\|\, \tilde{vk}_u \,\|\, g.\mathsf{gpk}$
If $\tilde{vk}_u \neq vk_u$ then return $\perp$
If $\neg\mathsf{DS.Ver}(vk_u, m_{\mathsf{DS}}, s)$ then return $\perp$
Return m

Fig. 14: Scheme SesLG-StE = Ses-SIGN-THEN-ENCRYPT[NE, DS] modelling symmetric signcryption in Session.

Construction 2. *Let* DS *be a digital signature scheme. Let* NE *be a nonce-based encryption scheme with associated data space* $\mathsf{NE.AD} = \{\varepsilon\}$. *Then* SesLG-StE = Ses-SIGN-THEN-ENCRYPT[NE, DS] *is the symmetric signcryption scheme as defined in Fig. 14, with group-key length* $\mathsf{SesLG\text{-}StE.gkl} = \mathsf{NE.kl}$, *nonce space* $\mathsf{SesLG\text{-}StE.NS} = \{0,1\}^{\mathsf{NE.nl}}$, *message space* $\mathsf{SesLG\text{-}StE.MS} = \{0,1\}^*$, *and associated data space* $\mathsf{SesLG\text{-}StE.AD} = \{\varepsilon\}$.

Security Analysis of SesLG-StE We now present our analysis of the IUF and OAE security of SesLG-StE. We identify three attacks, two against the IUF and one against the OAE security of SesLG-StE, that result from insufficient binding. One of the attacks against the IUF security and the attack against the OAE security are practically exploitable, and the broader protocol mitigates the second IUF security attack. We show that SesLG-StE achieves weaker versions of IUF and OAE security. In particular, we reduce the (weakened) IUF security of SesLG-StE to the SUFCMA security of DS and the (weakened) OAE security of SesLG-StE to the AE security of NE.

Attacks against IUF and OAE security of SesLG-StE. Consider the IUF security of SesLG-StE for a symmetric signcryption group $g = (gpk, epk)$ and a user u. An SesLG-StE ciphertext c encrypts $m \,\|\, vk_u \,\|\, s$ using the key K_g and the nonce n where s is a signature over $m \,\|\, vk_u \,\|\, gpk$. Notably, the signature s does not authenticate the ephemeral public key epk or the nonce n. The former implies that the signature does not fully bind to the symmetric signcryption group key identifier g. So SesLG-StE is vulnerable to variants Attack-1 and Attack-2 described in Section 3.2. [12] We refer to these attacks as Session-Attack-1 and Session-Attack-2 respectively.

Now consider the OAE security of SesLG-StE for a symmetric signcryption group $g = (gpk, epk)$ and a user u. In Section 3.2, we noted that in order to achieve optimal OAE security, the symmetric ciphertext produced by a symmetric signcryption scheme must bind to the sender identifier u. We can see from Fig. 14 that SesLG-StE never uses its input u. So the ciphertext c does not explicitly bind to the identifier of the sender. As a result, SesLG-StE is vulnerable to a variant of Attack-3 discussed in Section 3.2. We refer to this attack as Session-Attack-3.

Practical attacks and mitigations. We discuss the real-world implications of the attacks we described. The lack of binding in SesLG-StE is analogous to those identified in MLS-StE, but the practical consequences are different.

Session-Attack-1 **translates into an insider replay attack against Session.** Session-Attack-1 exploits the lack of binding between s and g. Specifically, the signature s does not authenticate the

[12] The XSalsa20-Poly1305 encryption scheme used to instantiate SesLG-StE in Session is not secure when a nonce is reused with a given symmetric key. However, our security game disallows the adversary from winning by nonce repetition. In Session, nonces are derived by applying the Blake2b hash function to $epk \,\|\, gpk$. This makes collisions unlikely as long as good randomness is used to generate epk.

ephemeral public key epk. In Session, all group members know gsk and epk. If c is the ciphertext for the symmetric signcryption group $g = (gpk, epk)$ sent by an honest member u, then a malicious member u' can decrypt c to retrieve $m \,\|\, vk_u \,\|\, s$ and re-encrypt it with $K_{g'}$ and nonce n' corresponding to $g' = (epk', esk')$. Member u' then broadcasts c' to the group, impersonating u. This attack allows the malicious member to indefinitely replay messages from other group members.[13]

The use of SesLG-StE in Session mitigates Session-Attack-2. The attack arises from the absence of binding between s and n. It is not exploitable due to the composition of SesLG-StE with the Blake2b hash function. In Session, group members deterministically derive nonces from $g = (gpk, epk)$, ensuring that the corresponding key K_g can be associated with at most one nonce. If a malicious member attempts to replay previously encrypted messages using a different nonce, receiving members can detect the forgery.

The exploitability of Session-Attack-3 depends on the broader implementation. The attack exploits the lack of explicit binding between c and u. In Session, a user's Account ID (or username) is the same as their verification key vk_u by default. However, users can opt to register a human-readable username through the Oxen Name System (ONS), which maps usernames to Account IDs and stores this mapping on the Arbitrum blockchain. In the default case, Session-Attack-3 is not exploitable as c and u are implicitly bound via vk_u. However, its exploitability when usernames are registered via ONS depends on the implementation of ONS, which we did not analyze.

Proving the weak IUF security of SesLG-StE. To account for weaknesses in the SesLG-StE scheme, we relax IUF security by using $\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}$ defined during the IUF security analysis of SesLG-StE. The security result (proved in Appendix C) follows similarly to the corresponding result for MLS-StE.

Theorem 3. *Let* $\mathsf{SesLG\text{-}StE} = \mathsf{Ses\text{-}SIGN\text{-}THEN\text{-}ENCRYPT}[\mathsf{NE}, \mathsf{DS}]$ *be the symmetric signcryption scheme built from some* NE *and* DS *as specified in Construction 2. Let* $\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 12. Let* $\mathcal{A}_{\mathsf{IUF}}$ *be any adversary against the* IUF *security of* SesLG-StE *with respect to* $\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}$*. Then we can build an adversary* $\mathcal{A}_{\mathsf{DS}}$ *such that*

$$\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{SesLG\text{-}StE},\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{DS}}).$$

Proving weak OAE security of SesLG-StE. We relax OAE security by using $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$. The formal result for the OAE security of SesLG-StE (proved in Appendix C) follows similarly to the corresponding result for MLS-StE.

Theorem 4. *Let* $\mathsf{SesLG\text{-}StE} = \mathsf{Ses\text{-}SIGN\text{-}THEN\text{-}ENCRYPT}[\mathsf{DS}, \mathsf{NE}]$ *be the symmetric signcryption scheme built from some* NE *and* DS *as specified in Construction 2. Let* $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 5. Let* $\mathcal{A}_{\mathsf{OAE}}$ *be any adversary against the* OAE *security of* SesLG-StE *with respect to* $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$*. Then we can build an adversary* $\mathcal{A}_{\mathsf{AE}}$ *such that*

$$\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SesLG\text{-}StE},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}}) \leq \mathsf{Adv}^{\mathsf{AE}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{NE}}).$$

We prove Theorem 3 and Theorem 4 in Appendix C.

Discussion. The OAE security of a scheme ensures that an attacker cannot forge ciphertexts without knowing the symmetric group key, even if they know the signing key(s) of group member(s). One needs to be careful when trying to understand how this guarantee lifts to a protocol — in this case Session — as a whole. We note, for example, that seemingly an attacker who knows the group's public key gpk (without knowing gsk) and an insider's signing key can forge messages to the group by creating a new group symmetric key from a fresh ephemeral esk. Mitigating this would require that gpk is only known to group members, an assumption that doesn't currently hold in Session.

However, Session stated the following in a public blog post, discussing why they do not aim for privacy when long-term keys are compromised.

[13] Session groups are additionally vulnerable to replays by group-outsiders. Session's implementation attaches a timestamp to the ciphertext c and uses it to detect replays. The timestamp is not authenticated by the symmetric ciphertext or the signature, allowing an outsider to modify timestamps and replay them to the group.

> [Perfect forward secrecy] means that if long-term keys for a given conversation are compromised, only a small amount of recent messages can be decrypted. However, under typical circumstances, ... the only way long term keys can be compromised is through full physical device access — in which case an attacker could simply pull the already-decrypted messages from the local database. As is often said in the infosec community, physical access is total access.

They were not referring to this specific attack in their blog post, but would presumably consider it similarly.

Disclosure. On February 10, 2025, we reached out to Session and shared our findings. They confirmed that Session-Attack-1 is possible in Legacy Groups but noted that weak mitigations exist, allowing detection of replays to some extent.

> ... clients provide a message info screen for every message, allowing users to see when a message was received versus when it was sent. This feature offers some detectability for such an attack. For example, if a message was "received" today but was "sent" two weeks ago, it could indicate that the timestamp was altered. However, detecting this can be challenging if a client is linking a new device or restoring from a recovery password.

They stated that Session-Attack-1 is resolved in GroupsV2 because a sender-generated timestamp is included as part of the input to the signature scheme which the receiver uses to order messages and detect replays. We have not analyzed GroupsV2 and hence did not verify this claim.

Regarding Session-Attack-3, they clarified aspects of their system that led us to update our earlier discussions.

They acknowledged that the attack we described in Section 4.2 is "partially possible" in Legacy Groups, highlighting the natural observation that the attacker doesn't know the underlying message in this replay attack. They also pointed to weak mitigations against this attack:

> Second, the decentralized node network used to store messages in Session deduplicates messages based on the message hash. Since the timestamp is not considered as part of the message hash by Session Nodes, an attacker cannot replay a message that is currently stored in the [node network]. This means the attacker would need to scrape messages and then replay them only after they had been deleted from the swarm, which typically occurs after 14 days.

Finally, they stated that additional mitigations are implemented in GroupsV2. We have not verified these claims.

5 Analysis of Encrypt-then-Sign Constructions

In this section, we analyze the chat encryption algorithms underlying the group messaging protocols in Signal and Matrix. Both Signal and Matrix implement variants of the Sender Keys [Mar14] protocol. We begin by providing a brief overview of the protocol. Then we model the chat encryption in Signal and Matirx as symmetric signcryption schemes and analyze their security.

The Sender Keys Protocol. The Sender Keys protocol is widely used by applications like Signal, Matrix, and WhatsApp[14] for end-to-end encrypted group chats. We summarize the protocol in Fig. 15. When a user u joins a group G, they generate a chain key ck (their symmetric key), a signing key pair (sk_u, vk_u), and initialize a counter (called the chain key counter) i to 0. The user u then creates a sender key object (ck, vk_u, i), consisting of the chain key, verification key, and chain key counter and shares it with other group members over two-party channels (typically instantiated using the Double Ratchet [MP16] algorithm). Every group member stores the sender keys of all group members as $\mathsf{senderKey}[G, u, e, i] = (\mathsf{ck}, vk)$, where each entry is indexed by the group identifier G, sender identifier u, epoch e, and chain key

[14] WhatsApp does not provide a precise description for its group chat encryption. Hence, we omit it from this work.

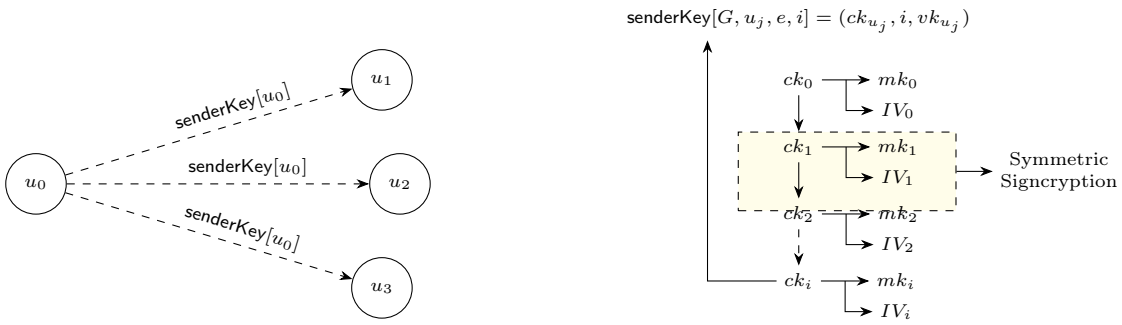


Fig. 15: An overview of the Sender Keys protocol for a group with four members u_j for $j \in [0, 3]$. **Left pane:** When u_1 joins the group, it creates and shares its sender key object with all other group members over 2-party channels. We use dashed lines to depict secure 2-party channels. **Right pane:** Each member stores the sender key of all group members and uses the chain key to derive the message key and IV, propagating the chain after every encryption/decryption.

SignalGroupEncrypt

$k_{\mathsf{seed}} \leftarrow \mathsf{HMAC\text{-}SHA\text{-}256}(\mathsf{ck}, 0\text{x}01)$ // $|k_{\mathsf{seed}}| = 256$

$k_{\mathsf{derived}} \leftarrow \mathsf{HKDF.Exp}(k_{\mathsf{seed}}, \text{“WhisperGroup”}, 384)$ // $|k_{\mathsf{derived}}| = 384$

$K \,\|\, IV \leftarrow k_{\mathsf{derived}}$ // s.t. $|K| = 256, |IV| = 128$

$c_{\mathsf{Enc}} \leftarrow \mathsf{CBC[AES\text{-}256].Enc}(K, IV, m)$

$c_{\mathsf{Sig}} \leftarrow \langle ad, G, \mathsf{chainId}, \mathsf{iteration}, c_{\mathsf{Enc}} \rangle$

$s \leftarrow\!\!\$\ \mathsf{XEdDSA.Sig}(sk_u, c_{\mathsf{Sig}})$

Fig. 16: The SignalGroupEncrypt subroutine.

index i. The epoch tracks the state of the group and is initialized to 0 during group creation, incrementing each time a member is added or removed or refreshes their sender key.

To send a message, a sender u first derives a message key mk and an initialization vector IV by applying a key derivation function to their chain key. The sender encrypts the message using mk and IV and a symmetric encryption scheme, and authenticates the resulting ciphertext using a digital signature scheme and their signing key sk_u. Finally, the sender broadcasts the ciphertext and signature to the group, derives a new chain key using a key derivation function and the current chain key, and increments i by 1. Group members track other members' sender keys to decrypt and verify incoming ciphertexts, replacing the sender's chain key with an appropriately derived key after decryption.

Assuming message keys are independently sampled, each message encryption can be analyzed in a setting where every member in the group owns a signing key pair, and all group members share a symmetric encryption key, allowing Sender Keys to be viewed as a series of single-use symmetric signcryption schemes.

5.1 Signal

As we mentioned at the beginning of this section, Signal uses the Sender Keys protocol to manage dynamic group membership and derive shared secrets such that each message sent within the group is encrypted using a fresh symmetric key. We begin by providing an overview of Signal's implementation of Sender Keys. In Signal, every group is assigned a 16-byte uniformly random identifier G and each user-device pair is assigned a unique identifier u. Users in Signal may have multiple devices, each uniquely identified by a globally unique user identifier and a device-specific identifier. For simplicity and without loss of applicability, we model each device in a group as an individual user with a distinct identifier u. When a user initiates a group G with members $u_0, u_1, ..., u_n$, each member $u_i \in G$ creates their sender key object as follows. First, they sample their ck, a uniformly random 256-bit string, and generate a fresh XEdDSA

Sig-EtS.SigEnc$(g, K_g, u, sk_u, n, m, ad)$	Sig-EtS.UserKg
// $K_g \in \{0,1\}^{256}$; $n = \{0,1\}^{128}$	$(sk, vk) \leftarrow\!\!\$$ DS.Kg ; Return (sk, vk)
$c_{\mathsf{NE}} \leftarrow$ NE.Enc(K_g, n, m)	Sig-EtS.VerDec$(g, K_g, u, sk_u, n, c, ad)$
// NE = CBC[AES-256]	$(c_{\mathsf{NE}}, s) \leftarrow c$
$c_{\mathsf{DS}} \leftarrow \langle ad, g, c_{\mathsf{NE}} \rangle$	$c_{\mathsf{DS}} \leftarrow \langle ad, g, c_{\mathsf{NE}} \rangle$
$s \leftarrow\!\!\$$ DS.Sig(sk_u, c_{DS})	If $\neg$DS.Ver$(vk_u, c_{\mathsf{DS}}, s)$ then return $\perp$
// DS = XEdDSA	$m \leftarrow$ NE.Dec$(K_g, n, c_{\mathsf{NE}})$
Return (c_{NE}, s)	Return m

Fig. 17: Scheme Sig-EtS = Signal-ENCRYPT-THEN-SIGN[NE, DS] modelling symmetric signcryption in Signal.

signing key pair (sk_u, vk_u). Next, they generate a uniformly random 31-bit integer called the chainId and initialize the index iteration for their sender key chain to 0. Within a group, the chainId and the iteration serve as a handle that group members use to retrieve a given user's current chain key.

Each group member u_i then shares their sender key object senderKey $= (G, \mathsf{chainId}, \mathsf{iteration}, \mathsf{ck}, vk_{u_i})$ with every other group member $u_j \in G \setminus u_i$. over one-on-one Signal sessions. All members in the group locally store the sender keys of other members using the senderKeyStore table. The senderKeyStore table is indexed by the tuple (G, u) and stores the corresponding senderKey object. When a new member is added to the group, existing members distribute their most recent sender key object to the newcomer. If a member is removed from the group, the remaining members generate fresh sender key objects and share them with one another. Finally, when a member resets or updates their sender key object, they propagate the update to all other group members. When user u with chain ID chainId sends the ith message in G, they use their current ck to deterministically derive the encryption key. After encrypting the message, they increment the iteration index by 1 and replace ck with a newly derived ck.

Chat encryption in Signal. Signal composes the Sender Keys protocol with the SignalGroupEncrypt subroutine shown in Fig. 16. It is an Encrypt-then-Sign construction and uses HMAC-SHA-256 hash function, the HKDF key derivation function, the AES-256 encryption scheme in the CBC mode, and the XEdDSA signature scheme as building blocks.

To send a message m with associated data ad from a user u to the group G, the SignalGroupEncrypt subroutine performs the following operations. First it derives the 256-bit seed key k_{seed} by applying the HMAC-SHA-256 hash function to the user's current chain key ck using the label 0x01. Then it expands k_{seed} into the 384-bit key k_{derived} by applying the HKDF expand function to k_{seed} using the label "WhisperGroup". It parses k_{derived} to obtain the 256-bit encryption key K and the 128-bit initialization vector IV. It calls the CBC[AES-256].Enc method to encrypt the message m using K and IV and obtains the ciphertext c_{Enc}. It builds the signature input c_{Sig} by uniquely encoding $ad, G,$ chainId, iteration and c_{Enc}. Then it invokes XEdDSA.Sig to sign c_{Sig} using the sender's group-specific signing key sk_u and obtains the signature s, and broadcasts (c_{Enc}, s) to the group.

Symmetric Signcryption in Signal. The chain key and the per-message encryption key derived from it are uniquely identified by $(G,$ chainId, iteration$)$. We use this to define symmetric signcryption groups that capture the single-use symmetric signcryption schemes. We define a symmetric signcryption group g as $g = (g.\mathsf{Id}, g.\mathsf{chainId}, g.\mathsf{iteration})$ where $g.\mathsf{Id} = G, g.\mathsf{chainId} = \mathsf{chainId}$, and $g.\mathsf{iteration} = \mathsf{iteration}$. For each symmetric signcryption group g, group members share a symmetric key $k_g = K$. Our Sig-EtS construction uses symmetric signcryption groups to model SignalGroupEncrypt as a symmetric signcryption scheme.

Construction 3. *Let* NE *be a random-IV-based encryption scheme and* DS *be a digital signature scheme. Then* Sig-EtS = Signal-ENCRYPT-THEN-SIGN[NE, DS] *is the symmetric signcryption scheme as defined in Fig. 17, with group-key length* Sig-EtS.gkl = NE.kl*, nonce space* Sig-EtS.NS $= \{0,1\}^{\mathsf{NE.nl}}$*, message space* Sig-EtS.MS $= \{0,1\}^*$*, and associated data space* Sig-EtS.AD $= \{0,1\}^*$.

Security Analysis of Sig-EtS We identify three attacks, one against the IUF security of Sig-EtS and two against the OAE security of Sig-EtS. The first results from the insufficient binding between s and n.

The second arises from insufficient binding between the symmetric ciphertext c_{Enc} and u, and the third from the use of an encryption scheme that lacks ciphertext integrity. The first and second attacks are mitigated by implicit binding, the third attack is practically exploitable. We show that Sig-EtS achieves weaker versions of IUF and OAE security. For the latter, we restrict the adversary from corrupting any users.

Attacks against the IUF and OAE security of Sig-EtS. Consider the IUF security of Sig-EtS for a symmetric signcryption group g and a user u. From Fig. 17 we can see that s does not authenticate n, making it suscptible to variants of Attack-2 defined in Section 3.2. We refer to this as Signal-Attack-1.

Consider the OAE security of Sig-EtS for a group $g = (G, \mathsf{chainId}, \mathsf{iteration})$ and a user u. The CBC[AES-256] encryption scheme does not provide integrity of ciphertexts. This allows group-outsider to modify an honestly generated ciphertext so that it decrypts to a message different from the original plaintext. The attack is as follows. The adversary exposes an honest user u that belongs to a group g and learns its signing key sk_u. It queries its SigEnc oracle on (g, u, n, m_0, m_1, ad) for some n, m_0, and m_1 where $m_0 \neq m_1$ to receive the ciphertext (c_{NE}, s). It modifies c_{NE} to obtain c'_{NE} and signs c'_{NE} using the exposed signing key $\mathsf{sk}[u]$ to produce the signature s' on $\langle ad, g, c'_{\mathsf{NE}}\rangle$. Finally, it queries (c'_{NE}, s') to VerDec and wins its game by returning 1 whenever VerDec returns 1. For the remainder of the section, we refer to this attack as Signal-Attack-2.

Moreover, we can see from Fig. 17 that Sig-EtS never uses its input u. So the ciphertext c does not explicitly bind to the identifier of the sender, making Sig-EtS is vulnerable to a variant of Attack-3 discussed in Section 3.2. We refer to this attack as Signal-Attack-3.

Practical attacks and mitigations. We discuss the real-world implications of Signal-Attack-1 and Signal-Attack-2. **Implicit binding between s and n prevents Signal-Attack-1.** The symmetric signcryption group key identifier $g = (G, \mathsf{chainId}, \mathsf{iteration})$ uniquely defines a symmetric key K_g and a nonce n. Users in Signal use g to deterministically derive n. Therefore, binding the signature to g implicitly also binds it to n, preventing Signal-Attack-1.

Signal-Attack-2 **leads into a potential malleability attack against Signal.** Signal-Attack-1 results from the lack of integrity of ciphertexts produced by CBC[AES-256]. We describe how a practical implementation of this attack would proceed, though we admit that the setting we describe is narrow. Say an attacker compromises the signing key sk_u of a user u. Thereafter, the user u joins a group G. The attacker can then maul ciphertexts (like xor-ing a ciphertext block with some bit-string $\tilde{m}$) produced by u and re-sign them using sk_u, thereby impersonating u to members of G.

The implicit binding between c and u prevents Signal-Attack-3. The chainId uniquely identifies a user within a group up to collisions. The symmetric signcryption group key identifier g contains the chainId and hence uniquely identifies a user u in the group. Since the ciphertext c_{NE} is produced using K_g, it implicitly binds to the corresponding user u, rendering Signal-Attack-2 not exploitable.

Proving weak IUF security of Sig-EtS. We relax the IUF security definition for Sig-EtS to account for Signal-Attack-1. A natural way would be to use the $\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}$ predicate. However, Sig-EtS does not achieve even this weaker notion.So we relax the definition by restricting to a class of adversaries that only make one SigEnc query per group g, and always query "matching" nonces to SigEnc and VerDec for a given group. Then we show that the IUF security Sig-EtS with respect to the $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ predicate reduces to the SUFCMA security of the underlying digital signature scheme.

Theorem 5. *Let* Sig-EtS = Signal-ENCRYPT-THEN-SIGN[NE, DS] *be the symmetric signcryption scheme built from some* NE *and* DS *as specified in Construction 3. Let* $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 5. Let* $\mathcal{A}_{\mathsf{IUF}}$ *be an adversary against the* IUF *security of* Sig-EtS *with respect to* $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ *that for each group g makes exactly one query to* SigEnc*, and queries the same nonce to* SigEnc *and* VerDec*. Then we can build an adversary* $\mathcal{A}_{\mathsf{SUFCMA}}$ *such that*

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

Proving weak OAE security of Sig-EtS. We relax the OAE game by disallowing the adversary from corrupting users. We show that Sig-EtS is OAE secure as long as the attacker does not learn any secrets for the group it is attacking.

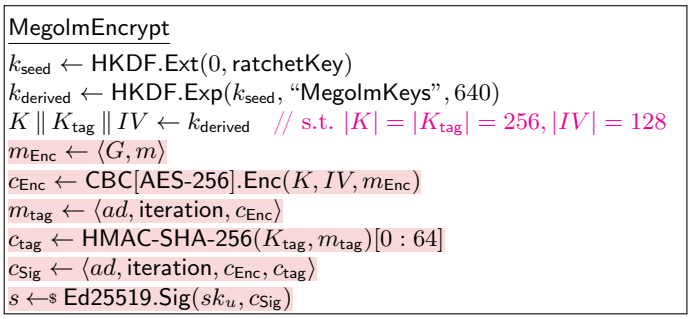


Fig. 18: The MegolmEncrypt subroutine used in Signal to encrypt messages from a user to a group.

We briefly sketch our result. The adversary can either win by forging a ciphertext to VerDec or by breaking the left-or-right security of ciphertexts produced by SigEnc. To forge a ciphertext $c = (c_{\mathsf{NE}}, s)$, the adversary must either forge c_{NE} or s. If the attacker forges c_{NE}, then it must also forge a corresponding signature. The SUFCMA security of DS prevents this. Now the attacker can only win by learning the bit b from left-or-right encryptions produced by SigEnc. The IND-CPA security of NE prevents this.

Theorem 6. *Let* $\mathsf{Sig\text{-}EtS} = \mathsf{Signal\text{-}ENCRYPT\text{-}THEN\text{-}SIGN}[\mathsf{NE}, \mathsf{DS}]$ *be the symmetric signcryption scheme built from some* NE *and* DS *as specified in Construction 3. Let* $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 5. Let* $\mathbb{A}^{\mathsf{SS}}_{\$\text{-}\mathsf{IV}}$ *and* $\mathbb{A}^{\mathsf{NE}}_{\$\text{-}\mathsf{IV}}$ *be the classes of adversaries defined in Section 2 and Section 3 respectively. Let* $\mathcal{A}_{\mathsf{OAE}} \in \mathbb{A}^{\mathsf{SS}}_{\$\text{-}\mathsf{IV}}$ *be an adversary against the* OAE *security of* Sig-EtS *with respect to* $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ *that never queries its* ExposeUser *or* NewCorrUser *oracles. Then we can build an adversaries* $\mathcal{A}_{\mathsf{SUFCMA}}$ *and* $\mathcal{A}_{\mathsf{IND\text{-}CPA}} \in \mathbb{A}^{\mathsf{NE}}_{\$\text{-}\mathsf{IV}}$ *such that*

$$\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{Sig\text{-}EtS},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}}) \leq 2 \cdot \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}}) + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{IND\text{-}CPA}}).$$

We prove Theorem 5 and Theorem 6 in Appendix D.

5.2 Matrix

In this section, we present our analysis of the symmetric signcryption scheme in Matrix. We begin with how Matrix's implementation, Megolm [Matb], of the Sender Keys protocol differs from our description at the beginning of this section. Each user u has two long-term key pairs: an Ed25519 signing key pair (SK_u, VK_u) called the *fingerprint key pair* and a Curve25519 encryption key pair (EK_u, DK_u) called the *identity key pair*. Users sign their identity public key EK_u using their long-term signing key SK_u, and publish the signature, VK_u, and EK_u to the Matrix network. It defines the sender key object as $(\mathsf{ck}, EK_u, vk_u, \mathsf{iteration})$. A sender's current chain key in a group is uniquely determined by $(G, EK_u, vk_u, \mathsf{iteration})$.

Chat encryption in Matrix. Megolm uses the MegolmEncrypt shown in Fig. 18 for chat encryption. It is an Encrypt-then-Sign construction and uses the HKDF [Kra10, KE10] key derivation function, the AES-256 encryption scheme in the CBC mode, the HMAC-SHA-256 [BCK96] message authentication scheme, and the Ed25519 signature scheme as building blocks.

To send a message m with associated data ad from a user u to the group G, MegolmEncrypt first derives a 640-bit key k_{derived} by applying the HKDF.Ext function to the sender's current chain key ck, with a salt of 0 and the label "MegolmKeys". It parses the key k_{derived} into three components: a 256-bit encryption key K, a 256-bit MAC key K_{tag}, and a 128-bit initialization vector IV. It encrypts the message m with the CBC[AES-256].Enc function using K and IV, producing the ciphertext c_{Enc}. Then it authenticates ad, iteration, and c_{Enc} using the HMAC-SHA-256 function and K_{tag}. It builds the input c_{Sig} to the signature scheme by encoding ad, iteration, c_{Enc}, and c_{tag}. It uses the Ed25519.Sig function to sign c_{Sig} with the sender's signing key sk_u, producing the signature s, and broadcasts the $((c_{\mathsf{Enc}}, c_{\mathsf{tag}}), s)$.

Mat-EtS.UserKg	Mat-EtS.VerDec$(g, K_g, u, sk_u, n, c, ad)$
$(sk, vk) \leftarrow\!\!\$$ DS.Kg ; Return (sk, vk)	$(g.\mathsf{Id}, g.\mathsf{ipk}, g.\mathsf{uvk}, g.\mathsf{iteration}) \leftarrow g$
Mat-EtS.SigEnc$(g, K_g, u, sk_u, n, m, ad)$	$(c_{\mathsf{NE}}, s) \leftarrow c$; $ad_{\mathsf{NE}} \leftarrow \langle ad, g.\mathsf{iteration}\rangle$
// $K_g \in \{0,1\}^{512}, n \in \{0,1\}^{128}$	$c_{\mathsf{DS}} \leftarrow \langle ad_{\mathsf{NE}}, c_{\mathsf{NE}}\rangle$
$(g.\mathsf{Id}, g.\mathsf{ipk}, g.\mathsf{uvk}, g.\mathsf{iteration}) \leftarrow g$	If $\neg$DS.Ver$(vk_u, c_{\mathsf{DS}}, s)$ then return $\perp$
$m_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, m\rangle$; $ad_{\mathsf{NE}} \leftarrow \langle ad, g.\mathsf{iteration}\rangle$	$m_{\mathsf{NE}} \leftarrow$ NE.Dec$(K_g, n, c_{\mathsf{NE}}, ad_{\mathsf{NE}})$
$c_{\mathsf{NE}} \leftarrow$ NE.Enc$(K_g, n, m_{\mathsf{NE}}, ad_{\mathsf{NE}})$	If $m_{\mathsf{NE}} = \perp$then return $\perp$
// NE = CBC[AES-256]-HMAC-SHA-256	$\langle G, m\rangle \leftarrow m_{\mathsf{NE}}$
$c_{\mathsf{DS}} \leftarrow \langle ad_{\mathsf{NE}}, c_{\mathsf{NE}}\rangle$	If $G \neq g.\mathsf{Id}$ then return $\perp$
$s \leftarrow\!\!\$$ DS.Sig(sk_u, c_{DS})	Return m
// DS = Ed25519	
Return (c_{NE}, s)	

Fig. 19: Symmetric signcryption scheme Mat-EtS = Mat-ENCRYPT-THEN-SIGN[NE, DS].

Symmetric Signcryption in Matrix. The chain key corresponding to a user u at index iteration in a group G is uniquely identified by $(G, EK_u, vk_u, \mathsf{iteration})$. Since k_{derived} is deterministically derived from the chain key and the per-message encryption key from k_{derived}, the latter is also uniquely identified by $(G, EK_u, vk_u, \mathsf{iteration})$. We use this observation to define a key group g as $g = (g.\mathsf{Id}, g.\mathsf{ipk}, g.\mathsf{uvk}, g.\mathsf{iteration})$ where $g.\mathsf{Id} = G, g.\mathsf{ipk} = EK_u, g.\mathsf{uvk} = vk_u$, and $g.\mathsf{iteration} = \mathsf{iteration}$. For each key group $g = (G, EK_u, vk_u, \mathsf{iteration})$, group members share a symmetric key $k_g = k_{\mathsf{derived}}$. Our Mat-EtS construction uses this definition of key groups to model the MegolmEncrypt subroutine as a symmetric signcryption scheme.

Construction 4. *Let* NE *be a random-IV-based authenticated encryption scheme with* $\mathsf{NE.AD} = \{0,1\}^*$ *and* DS *be a digital signature scheme. Then* Mat-EtS = Mat-ENCRYPT-THEN-SIGN[NE, DS] *is the symmetric signcryption scheme as defined in Fig. 19, with group-key length* $\mathsf{Mat\text{-}EtS.gkl} = \mathsf{NE.kl}$*, nonce space* $\mathsf{Mat\text{-}EtS.NS} = \{0,1\}^{\mathsf{NE.nl}}$*, associated data space* $\mathsf{MTX.AD} = \{0,1\}^*$*, message space* $\mathsf{Mat\text{-}EtS.MS} = \{0,1\}^*$.

Security Analysis of Mat-EtS We identify two attacks, one against the IUF security and the other against the OAE security of Mat-EtS: the first arises from insufficient binding between the s and g, and the second results from the lack of binding between the symmetric ciphertext c_{NE} and u. Both these attacks are mitigated by the broader protocol and implicit binding.

Attacks against the IUF and OAE security of Mat-EtS. Consider the IUF security of Mat-EtS for a key group $g = (G, EK_u, vk_u, \mathsf{iteration})$ and a user u. We can see from Fig. 19 that Mat-EtS never uses its inputs $g.\mathsf{Id}, g.\mathsf{ipk}, g.\mathsf{uvk}$, or the nonce n. Therefore Mat-EtS is vulnerable to variants of Attack-1 and Attack-2 described in Section 3.2. For the remainder of this section, we refer to these as Matrix-Attack-1 and Matrix-Attack-2.

Now consider the OAE security of Mat-EtS for a key group g and a user u. We can see from Fig. 14 that c_{NE} does not explicitly bind to u. So Mat-EtS is vulnerable to a variant of Attack-3 discussed in Section 3.2. We refer to this attack as Matrix-Attack-3.

Mitigations in the broader protocol. We show that both Matrix-Attack-1 and Matrix-Attack-2 are mitigated in the broader protocol.

The use of Mat-EtS **within Matrix prevents** Matrix-Attack-1, 2**.** First we address the lack of binding between the signature s and G and EK_u. In Matrix, a user has distinct signing key pairs for each group, preventing the exploitation of the former, and the identity key EK_u is also unique to a user (up to collisions), preventing exploitation of the latter.

Exploiting the lack of nonce binding is mitigated because Matrix ensures that at most one nonce can be used with a given key group g.

The implicit binding between c **and** u **prevents** Matrix-Attack-3**.** A key group g uniquely identifies a user within a group (up to collisions in EK and vk). Since the ciphertext c_{NE} is produced using K_g, it implicitly binds to the corresponding user u, mitigating Matrix-Attack-3.

PROVING THE WEAK IUF AND OAE SECURITY OF Mat-EtS. In our IUF and OAE security results for Mat-EtS, we use predicates $\mathsf{pred}^{\text{suf-except-group}}_{\text{trivial}}$ and $\mathsf{pred}^{\text{suf-except-user}}_{\text{trivial}}$ to account for the lack of user binding and group binding that lead to attacks Matrix-Attack-1 and Matrix-Attack-2 respectively.

Theorem 7. *Let* $\mathsf{Mat\text{-}EtS} = \mathsf{Mat\text{-}ENCRYPT\text{-}THEN\text{-}SIGN}[\mathsf{NE}, \mathsf{DS}]$ *be the symmetric signcryption scheme built from some* NE *and* DS *as specified in Construction 4. Let* $\mathsf{pred}^{\text{suf-except-group}}_{\text{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 5. Let* $\mathcal{A}_{\mathsf{IUF}}$ *be any adversary against the* IUF *security of* Mat-EtS *with respect to* $\mathsf{pred}^{\text{suf-except-group}}_{\text{trivial}}$*. Then we can build an adversary* $\mathcal{A}_{\mathsf{SUFCMA}}$ *such that*

$$\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{Mat\text{-}EtS},\mathsf{pred}^{\text{suf-except-group}}_{\text{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}}).$$

Theorem 8. *Let* $\mathsf{Mat\text{-}EtS} = \mathsf{Mat\text{-}ENCRYPT\text{-}THEN\text{-}SIGN}[\mathsf{NE}, \mathsf{DS}]$ *be the symmetric signcryption scheme built from some* NE *and* DS *as specified in Construction 4. Let* $\mathsf{pred}^{\text{euf-except-user}}_{\text{trivial}}$ *be the ciphertext-triviality predicate as defined in Fig. 5. Let* $\mathbb{A}^{\mathsf{SS}}_{\$\text{-}\mathsf{IV}}$ *and* $\mathbb{A}^{\mathsf{NE}}_{\$\text{-}\mathsf{IV}}$ *be the classes of adversaries defined in Section 2 and Section 3 respectively. Let* $\mathcal{A}_{\mathsf{OAE}} \in \mathbb{A}^{\mathsf{SS}}_{\$\text{-}\mathsf{IV}}$ *be an adversary against the* OAE *security of* Mat-EtS *with respect to* $\mathsf{pred}^{\text{euf-except-user}}_{\text{trivial}}$*. Then we can build an adversary* $\mathcal{A}_{\mathsf{AEAD}} \in \mathbb{A}^{\mathsf{NE}}_{\$\text{-}\mathsf{IV}}$ *such that*

$$\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{Mat\text{-}EtS},\mathsf{pred}^{\text{suf-except-user}}_{\text{trivial}},\mathsf{func}^{\perp}_{\text{out}}}(\mathcal{A}_{\mathsf{OAE}}) \leq \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}}).$$

The various implementations of the Sender Keys protocol follow similar procedures for establishing and propagating symmetric keys but differ in how these keys are applied to chat encryption, as seen in Signal and Matrix. By solely focusing on the symmetric signcryption instances underlying these implementations, our analysis helps precisely distinguish the varying security levels provided by different implementations.

6 Conclusions

We analyzed the composition of symmetric encryption and digital signatures in MLS, Session, Signal, and Matrix using the symmetric signcryption primitive of [JKS24]. Isolating chat encryption in these applications using the symmetric signcryption primitive allowed us to systematically identify potential attacks against all four applications that resulted from the lack of explicit binding between the signing and encryption components. To check whether these attacks translate into practical exploits, we carefully analyzed the use of these constructions within the broader protocol, looking for implicit bindings that may be enforced by higher level primitives. We identified novel insider replay attacks against MLS and Session. Additionally, we observed the forgery attack against Signal that was previously identified by [BCG23], though our analysis was narrower in scope than theirs. The discovery of these attacks not only speaks to the value of the symmetric signcryption primitive, but also highlights the importance of formally defined insider security notions for secure group messaging protocols.

References

ACD19. Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 129–158. Springer, Heidelberg, May 2019. doi:10.1007/978-3-030-17653-2_5. 3

ACDJ23. M. R. Albrecht, S. Celi, B. Dowling, and D. Jones. Practically-exploitable cryptographic vulnerabilities in matrix. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 164–181, Los Alamitos, CA, USA, may 2023. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.10351027, doi:10.1109/SP46215.2023.10351027. 3, 6

ACDT20. Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 248–277. Springer, Heidelberg, August 2020. doi:10.1007/978-3-030-56784-2_9. 6

ACDT21. Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Modular design of secure group messaging protocols and the security of mls. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 1463–1483, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3460120.3484820. 3, 6, 17

ACJM20. Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. Continuous group key agreement with active security. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 261–290. Springer, Heidelberg, November 2020. doi:10.1007/978-3-030-64378-2_10. 6

ADJ24. M. R. Albrecht, B. Dowling, and D. Jones. Device-oriented group messaging: A formal cryptographic analysis of matrix' core. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 2666–1685, Los Alamitos, CA, USA, may 2024. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00075, doi:10.1109/SP54263.2024.00075. 3, 6

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. 4

AJM22. Joël Alwen, Daniel Jost, and Marta Mularczyk. On the insider security of MLS. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 34–68. Springer, Heidelberg, August 2022. doi:10.1007/978-3-031-15979-4_2. 6

AMPS22. Martin R. Albrecht, Lenka Mareková, Kenneth G. Paterson, and Igors Stepanovs. Four attacks and a proof for telegram. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 87–106, 2022. doi:10.1109/SP46214.2022.9833666. 3

AP24. Martin R. Albrecht and Kenneth G. Paterson. Analysing cryptography in the wild - a retrospective. Cryptology ePrint Archive, Paper 2024/532, 2024. URL: https://eprint.iacr.org/2024/532. 3

BBR18. Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. Treekem: Asynchronous decentralized key management for large dynamic groups a protocol proposal for messaging layer security (mls). https://hal.inria.fr/hal-02425247/file/treekem+%281%29.pdf, may 2018. 3

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. URL: https://www.rfc-editor.org/info/rfc9420, doi:10.17487/RFC9420. 3, 6, 13, 16, 17

BCG23. David Balbás, Daniel Collins, and Phillip Gajland. WhatsUpp with sender keys? Analysis, improvements and security proofs. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part V*, volume 14442 of *LNCS*, pages 307–341. Springer, Heidelberg, December 2023. doi:10.1007/978-981-99-8733-7_10. 3, 6, 27

BCK96. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 1–15. Springer, Heidelberg, August 1996. doi:10.1007/3-540-68697-5_1. 25

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. 18

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. 18

Ber06. Daniel J. Bernstein. Curve25519: new diffie-hellman speed records. In *Proceedings of the 9th International Conference on Theory and Practice of Public-Key Cryptography*, PKC'06, page 207–228, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11745853_14. 18

Ber08. Daniel J. Bernstein. *The Salsa20 Family of Stream Ciphers*, page 84–97. Springer-Verlag, Berlin, Heidelberg, 2008. URL: https://doi.org/10.1007/978-3-540-68351-3_8. 18

BFG+22. Alexander Bienstock, Jaiden Fairoze, Sanjam Garg, Pratyay Mukherjee, and Srinivasan Raghuraman. A more complete analysis of the Signal double ratchet algorithm. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 784–813. Springer, Heidelberg, August 2022. doi:10.1007/978-3-031-15802-5_27. 3

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. 30

BRO+24. Benjamin Beurdouche, Eric Rescorla, Emad Omara, Srinivas Inguva, and Alan Duric. The Messaging Layer Security (MLS) Architecture. Internet-Draft draft-ietf-mls-architecture-15, Internet Engineering Task Force, August 2024. Work in Progress. URL: https://datatracker.ietf.org/doc/draft-ietf-mls-architecture/15/. 6, 17

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. 3

CGCD+17. Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 451–466, 2017. doi:10.1109/EuroSP.2017.27. 3

CS03. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003. 18

DFG+23. Gareth T. Davies, Sebastian H. Faller, Kai Gellert, Tobias Handirk, Julia Hesse, Máté Horváth, and Tibor Jager. Security analysis of the WhatsApp end-to-end encrypted backup protocol. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 330–361. Springer, Heidelberg, August 2023. doi:10.1007/978-3-031-38551-3_11. 3

FG24. Rune Fiedler and Felix Günther. Security analysis of signal's PQXDH handshake. Cryptology ePrint Archive, Paper 2024/702, 2024. URL: https://eprint.iacr.org/2024/702. 3

JKS24. Joseph Jaeger, Akshaya Kumar, and Igors Stepanovs. Symmetric signcryption and E2EE group messaging in keybase. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part III*, volume 14653 of *LNCS*, pages 283–312. Springer, Heidelberg, May 2024. doi:10.1007/978-3-031-58734-4_10. 3, 27

JSH24. Kee Jefferys, Maxim Shishmarev, and Simon Harman. Session: End-to-end encrypted conversations with minimal metadata leakage, 2024. URL: https://arxiv.org/abs/2002.04609, arXiv:2002.04609. 18

KE10. Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, May 2010. URL: https://www.rfc-editor.org/info/rfc5869, doi:10.17487/RFC5869. 25

Kli21. Tijana Klimovic. Modular design of the messaging layer security (mls) protocol. Master's thesis, ETH Zurich, 2021. Accessed: 2024-09-28. URL: https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/appliedcrypto/education/theses/masters-thesis_tijana-klimovic.pdf. 17

KPPW+21. Karen Klein, Guillermo Pascual-Perez, Michael Walter, Chethan Kamath, Margarita Capretto, Miguel Cueto, Ilia Markov, Michelle Yeo, Joël Alwen, and Krzysztof Pietrzak. Keep the dirt: Tainted TreeKEM, adaptively and actively secure continuous group key agreement. In *2021 IEEE Symposium on Security and Privacy*, pages 268–284. IEEE Computer Society Press, May 2021. doi:10.1109/SP40001.2021.00035. 6

Kra10. Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, Heidelberg, August 2010. doi:10.1007/978-3-642-14623-7_34. 25

Mar14. Moxie Marlinspike. Private group messaging. https://signal.org/blog/private-groups/, may 2014. 3, 21

Mata. Matrix. Matrix. https://matrix.org/. 3

Matb. Matrix. Megolm group ratchet. https://gitlab.matrix.org/matrix-org/olm/-/raw/master/docs/megolm.md. 25

MP16. Moxie Marlinspike and Trevor Perrin. The double ratchet algorithm. https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf, nov 2016. 3, 21

RMS18. Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: On the end-to-end security of group chats in signal, whatsapp, and threema. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 415–429, 2018. doi:10.1109/EuroSP.2018.00036. 3

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. 7

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. 7

SA15. Markku-Juhani O. Saarinen and Jean-Philippe Aumasson. The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). RFC 7693, November 2015. URL: https://www.rfc-editor.org/info/rfc7693, doi:10.17487/RFC7693. 18

Sesa. Session. Session. https://www.getsession.org/. 3

Sesb. Session. Sessiondesktop – MessageEncrypter.ts – encryptUsingSessionProtocol. https://github.com/oxen-io/session-desktop/blob/unstable/ts/session/crypto/MessageEncrypter.ts#L65. 18

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. 7

Sig. Signal. Signal. https://signal.org/. 3

A Notation and Standard Definitions

A.1 Notation

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$. Let ε denote the empty string.. 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 $1 \le i \le j \le |x|$. We denote the concatenation of strings $x, y \in \{0,1\}^*$ by $x \,\|\, y$. 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 then $x \leftarrow\!\!\$\, X$ denotes sampling uniformly from X. If T is a table, $\mathsf{T}[i]$ denotes the element of the table that is indexed by i. We let $\perp \notin \{0,1\}^*$ be an error code that indicates rejection. When not specified, integers are initialized to 0, strings to $\perp$, booleans to false, sets to the empty set, and tables empty.

If $\mathcal{A}$ is an algorithm then $y \leftarrow\!\!\$\, \mathcal{A}^{\mathrm{O}_1, \mathrm{O}_2, \cdots}(x_1, x_2, x_3, \ldots)$ denotes providing $\mathcal{A}$ with oracle access to $\mathrm{O}_1, \mathrm{O}_2, \ldots$ (that it can call arbitrary number of times in any order) and running it on inputs $x_1, x_2, \ldots$ to produce output y. For deterministic $\mathcal{A}$ we use $\leftarrow$ instead of $\leftarrow\!\!\$$. $[\mathcal{A}(x)]$ denotes the set of all possible outputs when $\mathcal{A}$ is run on x. The $\mathbf{abort}(x)$ instruction immediately halts an algorithm and outputs x. Adversaries are algorithms and do not query their oracles on $\perp$.

For our definitions and proofs, we used the code-based framework of [BR06]. Security notions are defined using games like in **??**. If $\mathcal{G}$ is a game then $\Pr[\mathcal{G}]$ denotes the probability that the game outputs true. "require$x$" is shorthand for "If $\neg x$ then return $\perp$". We use a sequence of "hybrid" games within a single figure for our proofs. Comments of the form // $\mathcal{G}[i, j)$ indicate that a line of code is included in games $\mathcal{G}_k$ for $i \le k < j$. To identify changes made to the k-th hybrid, one should look for lines commented as // $\mathcal{G}[i, k)$ for code that is no longer part of the k-th hybrid and // $\mathcal{G}[k, j)$] for code that is newly introduced in the k-th hybrid.

We frequently use the fundamental lemma of game playing in our proofs [BR06]. Let $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ be games that are syntactically identical until after a boolean flag bad is set. Then,

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

where $\Pr[\mathsf{bad}^{\mathcal{G}_i}]$ denotes the probability that bad was set in $\mathcal{G}_i$.

A.2 Standard primitives

We use various standard cryptographic primitives including symmetric encryption schemes, nonce-based encryption schemes, and digital signature schemes.

Symmetric encryption. A symmetric encryption scheme SE specifies two algorithms $\mathsf{SE.Enc}$ and $\mathsf{SE.Dec}$. Additionally, SE specifies a key length $\mathsf{SE.kl} \in \mathbb{N}$. Then encryption algorithm $\mathsf{SE.Enc}$ takes a key $K \in \{0,1\}^{\mathsf{SE.kl}}$ and a message $m \in \{0,1\}^*$ and outputs a ciphertext c. The deterministic decryption algorithm $\mathsf{SE.Dec}$ takes a key K and a ciphertext c and outputs a message $m \in \{0,1\}^* \cup \{\perp\}$ where $\perp$ indicates decryption failure. The decryption correctness of a symmetric encryption scheme requires that $\mathsf{SE.Dec}(K, \mathsf{SE.Enc}(K, m)) = m$ holds for all $K \in \{0,1\}^{\mathsf{SE.kl}}$ and $m \in \{0,1\}^*$.

Nonce-based encryption. A nonce-based encryption scheme NE specifies two deterministic algorithms $\mathsf{NE.Enc}$ and $\mathsf{NE.Dec}$ along with 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 key $K \in \{0,1\}^{\mathsf{NE.kl}}$, a nonce $n \in \{0,1\}^{\mathsf{NE.nl}}$, a message $m \in \{0,1\}^*$, and associated data $ad \in \mathsf{NE.AD}$ and returns a ciphertext c. The decryption algorithm $\mathsf{NE.Dec}$ takes K, n, c, ad and outputs a message $m \in \{0,1\}^* \cup \{\perp\}$ where $\perp$ denotes decryption failure. The decryption correctness of a nonce-based encryption scheme requires that $\mathsf{NE.Dec}(K, n, \mathsf{NE.Enc}(K, n, m, ad), ad) = m$ holds for all $K \in \{0,1\}^{\mathsf{SE.kl}}$, $n \in \{0,1\}^{\mathsf{NE.nl}}$, $m \in \{0,1\}^*$, and $ad \in \mathsf{NE.AD}$. For nonce-based encryption schemes with $\mathsf{NE.AD} = \{\varepsilon\}$, we omit the last input from $\mathsf{NE}.enc$ and $\mathsf{NE}.dec$.

Digital signatures. A digital signature scheme DS specifies a signing key length $\mathsf{DS.skl} \in \mathbb{N}$, a signature length $\mathsf{DS.sl} \in \mathbb{N}$, and three algorithms — 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 $m \in \{0,1\}^*$ and returns a signature $s \in \{0,1\}^{\mathsf{DS.sl}}$, and the verification algorithm $\mathsf{DS.Ver}$ takes vk, m, and s an returns a boolean based on whether s is a valid signature on m under vk. The correctness of a digital signature scheme requires that $\mathsf{DS.Ver}(vk, m, s) = \mathsf{true}$ holds for all $(sk, vk) \in [\mathsf{DS.Kg}]$, all $m \in \{0,1\}^*$, and all $s \in [\mathsf{DS.Sig}(sk, m)]$.

B Proofs for MLS

B.1 In-group UnForgeability of MLS-StE (Proof of Theorem 1)

Games $\mathcal{G}_0$–$\mathcal{G}_2$

$\mathcal{A}_{\mathsf{IUF}}^{\mathrm{U,G,SigEnc,VerDec}}$; Return win

SigEnc(g, u, n, m, ad)

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

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 IUF game as per Fig. 4.

VerDec(g, u, n, c, ad)

require $\mathsf{K}[g] \neq \bot$ and $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$ad_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, ad\rangle$
$m_{\mathsf{NE}} \leftarrow \mathsf{NE.Dec}(K_g, n, c, ad_{\mathsf{NE}})$
If $m_{\mathsf{NE}} = \bot$ then return $\bot$
$m \,\|\, s \leftarrow m_{\mathsf{NE}}$
$m_{\mathsf{DS}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, g.\mathsf{leafIndex}, ad, m\rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], m_{\mathsf{DS}}, s)$ then return $\bot$
$z \leftarrow ((g, u, n, m, ad), c)$
If $\exists g', n', c' : ((g', u, n', m, ad), c') \in C$ then return m
If $\neg\mathsf{user_is_corrupt}[u]$ then
  If $(u, m_{\mathsf{DS}}, s) \notin S$ then
    $\mathsf{bad}_0 \leftarrow \mathsf{true}$ // $\mathcal{A}_{\mathsf{SUFCMA}}$ breaks SUFCMA of DS.
    win $\leftarrow$ true // $\mathcal{G}_{[0,1)}$
  Else // $(u, m_{\mathsf{DS}}, s) \in S$
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$ // This step is unreachable
    win $\leftarrow$ true // $\mathcal{G}_{[0,2)}$
Return m

Adversary $\mathcal{A}_{\mathsf{SUFCMA}}^{\mathrm{U,Sign}}$

$\mathcal{A}_{\mathsf{IUF}}^{\mathrm{SimU,SimG,SimSigEnc,SimVerDec}}$

SimSigEnc(g, u, n, m, ad)

require $\mathsf{K}[g] \neq \bot$ and $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$m_{\mathsf{DS}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, g.\mathsf{leafIndex}, ad, m\rangle$
$s \leftarrow\!\$\ \mathrm{Sign}(u, m_{\mathsf{DS}})$; $m_{\mathsf{NE}} \leftarrow m \,\|\, s$
$ad_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, ad\rangle$
$c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_{\mathsf{NE}}, ad_{\mathsf{NE}})$
$S \leftarrow S \cup \{(u, c_{\mathsf{DS}}, s)\}$
$C \leftarrow C \cup \{((g, u, n, m, ad), c)\}$; Return c

SimNewHonGroup(g, users)
SimExposeGroup(g)
SimNewCorrGroup(g, K, users)
// These oracles are identical to the corresponding
// oracles in the IUF game for MLS-StE as per Fig. 4.

SimVerDec(g, u, n, c, ad)

require $\mathsf{K}[g] \neq \bot$ and $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$ad_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, ad\rangle$
$m_{\mathsf{NE}} \leftarrow \mathsf{NE.Dec}(K_g, n, c, ad_{\mathsf{NE}})$
If $m_{\mathsf{NE}} = \bot$ then return $\bot$
$m \,\|\, s \leftarrow m_{\mathsf{NE}}$
$m_{\mathsf{DS}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, g.\mathsf{leafIndex}, ad, m\rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], m_{\mathsf{DS}}, s)$ then return $\bot$
$z \leftarrow ((g, u, n, m, ad), c)$
If $\exists g', n', c' : ((g', u, n', m, ad), c') \in C$ then return m
If $\neg\mathsf{user_is_corrupt}[u]$ then
  If $(u, c_{\mathsf{DS}}, s) \notin S$ then
    $z \leftarrow (u, c_{\mathsf{DS}}, s)$
    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 MLS-StE.SigEnc, MLS-StE.VerDec in game $\mathcal{G}^{\mathsf{IUF}}_{\mathsf{MLS\text{-}StE},\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}})$. The code added for the transitions between games is highlighted in purple. **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.

1. $\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{MLS\text{-}StE},\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) = \Pr[\mathcal{G}_0]$
2. $\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$
3. $\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \leq \Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$
4. $\Pr[\mathcal{G}_2] = 0$
5. $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})$
6. $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}] = 0$

By combining the above, we have

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

We justify our claims in the following paragraphs.

GAME $\mathcal{G}_0$. Game $\mathcal{G}_0$ is functionally equivalent to game $\mathcal{G}^{\mathsf{IUF}}_{\mathsf{MLS\text{-}StE},\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}})$. 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 purple 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

$$\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{MLS\text{-}StE},\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) = \Pr[\mathcal{G}_0].$$

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

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

Each of these transitions removes an instruction that sets the win flag. The win flag can no longer be set in game $\mathcal{G}_2$, 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 $\mathbf{abort}(u, m_s, s)$ whenever it detects bad_0 being set in the simulated IUF security game. We have

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

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 $\nexists g', n', c' \colon ((g', u, n', m, ad), c') \in C$ and $(u, m_s, s) \in S$ are simultaneously true. We can use the definition m_s to expand $(u, m_s, s) \in S$ into $((u, \langle g.\mathsf{id}, g.\mathsf{ep}, g.\mathsf{leafIndex}, ad, m\rangle), s) \in S$. Therefore, bad_1 is only set in $\mathcal{G}_1$ when $\nexists g', u', c' \colon ((g', u, n', m, ad), c') \in C$ and $((u, \langle g.\mathsf{id}, g.\mathsf{ep}, g.\mathsf{leafIndex}, ad, m\rangle), s) \in S$. The latter means that there was a previous call to SIGENC of the form (g, u, n, m, ad). Let c be the ciphertext that was returned by this SIGENC call. Then by definition of the SIGENC oracle we have that $((g, u, n, m, ad), c) \in C$. But the first condition gives us $\nexists g', u', c' \colon ((g', u, n', m, ad), c') \in C$. We get a contradiction, and hence

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

B.2 Out-group Authenticated Encryption of MLS-StE (Proof of Theorem 2)

Games $\mathcal{G}_0$–$\mathcal{G}_2$

$b \leftarrow\$ \{0,1\}$; $b' \leftarrow\$ \mathcal{A}_{\mathsf{OAE}}^{\mathrm{U,G,SigEnc,VerDec}}$; Return $b = b'$

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 $(g, n) \notin N$
If $m_0 \neq m_1$ then
  If $\mathsf{group_is_corrupt}[g]$ then return $\bot$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$m_{\mathsf{DS}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, g.\mathsf{leafIndex}, ad, m_b \rangle$
$s \leftarrow\$ \mathsf{DS.Sig}(\mathsf{sk}[u], m_{\mathsf{DS}})$; $m_{\mathsf{NE}} \leftarrow m_b \,\|\, s$
$ad_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, ad \rangle$
$c \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_{\mathsf{NE}}, ad_{\mathsf{NE}})$
$C \leftarrow C \cup \{((g, u, n, m_b, ad), c)\}$
$N \leftarrow N \cup \{(g, n)\}$
$W[g, u, n, c, ad] \leftarrow m_1$; Return c

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 return $\bot$ // $\mathcal{G}_{[1,\infty)}$
$ad_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, ad \rangle$
$m_{\mathsf{NE}} \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c, ad_{\mathsf{NE}})$
If $m_{\mathsf{NE}} = \bot$ then return $\bot$
$m \,\|\, s \leftarrow m_{\mathsf{NE}}$
$m_{\mathsf{DS}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, g.\mathsf{leafIndex}, ad, m \rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], m_{\mathsf{DS}}, s)$ then return $\bot$
$z \leftarrow ((g, u, n, m, ad), c)$ // $\mathcal{G}_{[0,2)}$
If $\exists u' : ((g, u', n, m, ad), c) \in C$ then return $\bot$ // $\mathcal{G}_{[0,2)}$
If $b = 0$ then return $\bot$ 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 MLS-StE as per Fig. 4.

Adversary $\mathcal{A}_{\mathsf{AEAD}}^{\mathrm{G,Enc,Dec}}$

$b' \leftarrow\$ \mathcal{A}_{\mathsf{OAE}}^{\mathrm{SimU,SimG,SimSigEnc,SimVerDec}}$; Return b'

SimSigEnc(g, u, n, m_0, m_1, ad)

require $\mathsf{K}[g] \neq \bot$ and $|m_0| = |m_1|$
require $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
If $m_0 \neq m_1$ then
  If $\mathsf{group_is_corrupt}[g]$ then return $\bot$ else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
  $\forall d \in \{0,1\}, m_{\mathsf{DS},d} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, g.\mathsf{leafIndex}, ad, m_d \rangle$
  $\forall d \in \{0,1\}, s_d \leftarrow\$ \mathsf{DS.Sig}(\mathsf{sk}[u], m_{\mathsf{DS},d})$
  $m_{\mathsf{NE},d} \leftarrow m_d \,\|\, s_d$
  $ad_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, ad \rangle$
  $c \leftarrow \mathrm{Enc}(\mathsf{K}[g], n, m_{\mathsf{NE},0}, m_{\mathsf{NE},1} ad_{\mathsf{NE}})$
Else // $m_0 = m_1$
  $m_s \leftarrow \langle g, n, m_0, ad \rangle$; $s \leftarrow\$ \mathsf{DS.Sig}(\mathsf{sk}[u], m_s)$
  $m_{\mathsf{NE}} \leftarrow m_0 \,\|\, s$; $ad_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, ad \rangle$
  $c \leftarrow \mathrm{Enc}((g, u), n, m_{\mathsf{NE}}, m_{\mathsf{NE}}, ad_{\mathsf{NE}})$
$W[g, u, n, c, ad] \leftarrow m_1$; 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 $\exists u' : W[g, u', n, c, ad] \neq \bot$ then return $\bot$
$ad_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, ad \rangle$
$m_{\mathsf{NE}} \leftarrow \mathrm{Dec}(K_g, n, c, ad_{\mathsf{NE}})$
If $m_{\mathsf{NE}} = \bot$ then return $\bot$
abort(1) // $m_{\mathsf{NE}} \neq \bot$ possible only when $b = 1$.

SimNewHonUser(u)
SimExposeUser(u)
SimNewCorrUser(u, sk, vk)
// These oracles are identical to the corresponding
// oracles in the OAE game for MLS-StE as per Fig. 4.

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

Fig. 21: **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 MLS-StE.SigEnc, MLS-StE.VerDec and the ciphertext-triviality predicate $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ in game $\mathcal{G}_{\mathsf{MLS\text{-}StE}, \mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}}^{\mathsf{OAE}}(\mathcal{A}_{\mathsf{OAE}})$. The code added for the transitions between games is highlighted in purple. **Bottom pane:** Adversary $\mathcal{A}_{\mathsf{AEAD}}$ for the proof of Theorem 2. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_2$.

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

1. $\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{MLS\text{-}StE},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1$
2. $\Pr[\mathcal{G}_0] = \Pr[\mathcal{G}_1]$
3. $\Pr[\mathcal{G}_1] = \Pr[\mathcal{G}_2]$
4. $2 \cdot \Pr[\mathcal{G}_2] - 1 \leq \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}})$

By combining the above, we have

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

We justify our claims in the following paragraphs.

Game $\mathcal{G}_0$. Game $\mathcal{G}_0$ is functionally equivalent to game $\mathcal{G}^{\mathsf{OAE}}_{\mathsf{MLS\text{-}StE},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}})$. 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 purple and does not affect its functionality; this code will be used for transitions between games. It follows that

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

Transition from $\mathcal{G}_0$ to $\mathcal{G}_1$. Game $\mathcal{G}_1$ maintains a table W within the SigEnc oracle, that maps (g, u, n, c, ad) to m_1, to record all ciphertexts that are returned by the SigEnc oracle. Subsequently $\mathcal{G}_1$ uses the table W in VerDec to detect if the adversary queries such a ciphertext to the VerDec oracle (with the matching values of g, n, ad), and immediately returns $\perp$. We claim that

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

To justify this claim, we look at how $\mathcal{G}_0$ treats an honestly produced ciphertext that was 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 C, and Q are updated to insert new elements that are created from the input $(g^*, u^*, n^*, m_0^*, m_1^*, ad^*)$ and output c^*. 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 $\exists u' \colon ((g, u', n, m, ad), c) \in C$ is true and the VerDec oracle returns $\perp$. 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 $\exists u' \colon ((g, u', n, m, ad), c) \in C$ in the VerDec oracle of game $\mathcal{G}_1$ can never be reached. In particular, if $\exists u' \colon ((g, u', n, m, ad), c) \in C$ evaluates to true then $\exists u' \colon W[g, u, n, c, ad] \neq \perp$ must also be true. The latter causes VerDec to return $\perp$ at the beginning of the oracle execution. So in game $\mathcal{G}_2$ we remove the two lines of code that build z and check whether it the query is a trivially forwarded ciphertext. Game $\mathcal{G}_2$ is 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 the 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 we require DS to produce signatures of a fixed length $\mathsf{DS.sl}$.

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 $m_e \neq \perp$ 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 $\mathbf{abort}(1)$ in order to halt and return 1 as output. Otherwise (i.e. $m_e = \perp$) adversary $\mathcal{A}_{\mathsf{AEAD}}$ returns $\perp$ as the output of the simulated VerDec oracle. Note that the AEAD security game defines Dec to return $\perp$ 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 $\perp$ if $W[g, u, n, c, ad] = \perp$ 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-$\perp$ 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.

C Proofs for Session

C.1 In-group UnForgeability of SesLG-StE (Proof of Theorem 3)

OVERVIEW. We use games $\mathcal{G}_0$ through $\mathcal{G}_2$ in Fig. 22 to establish the following claims.

1. $\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{SesLG\text{-}StE},\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) = \Pr[\mathcal{G}_0]$
2. $\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$
3. $\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \leq \Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$
4. $\Pr[\mathcal{G}_2] = 0$
5. $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})$
6. $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}] = 0$

By combining the above, we have

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

The sequence of hybrids we use for this proof mirrors those used to prove the IUF security of MLS-StE in Appendix B. So the justifications for the corresponding claims follow analogously from those for MLS-StE. The only difference is the justification for the bound on $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$ (claim 6). We justify this claim below. We do not provide individual descriptions to justify the other claims. Readers interested in understanding each transition can refer to the relevant explanations in the prior section while considering Fig. 22.

BOUNDING $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$. We know that $ad = \varepsilon$ always holds. The flag bad_1 is only set in game $\mathcal{G}_1$ when $\nexists g', n', c' \colon ((g', u, n', m, \varepsilon), c') \in C$ and $(u, m_s, s) \in S$ are simultaneously true. We can use the definition m_s to expand $(u, m_s, s) \in S$ into $(u, (m \,\|\, vk_u \,\|\, g.\mathsf{gpk}), s) \in S$. Therefore, bad_1 is only set in $\mathcal{G}_1$ when $\nexists g', n', c' \colon ((g', u, n', m, \varepsilon), c') \in C$ and $(u, (m \,\|\, vk_u \,\|\, g.\mathsf{gpk}), s) \in S$. The latter means that there was a previous call to SIGENC of the form $(\tilde{g}, u, \tilde{n}, m, \varepsilon)$ such that $\tilde{g}.\mathsf{gpk} = g.\mathsf{gpk}$. Let $\tilde{c}$ be the ciphertext that was returned by this SIGENC call. Then by definition of the SIGENC oracle we have that $((\tilde{g}, u, \tilde{n}, m, \varepsilon), \tilde{c}) \in C$. But the first condition gives us $\nexists g', n', c' \colon ((g', u, n', m, \varepsilon), c') \in C$. We get a contradiction, and hence

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

Games $\mathcal{G}_0$–$\mathcal{G}_2$

$\mathcal{A}_{\mathsf{IUF}}^{\mathrm{U,G,SigEnc,VerDec}}$; Return win

SigEnc(g, u, n, m, ad) // $ad = \varepsilon$

require $\mathsf{K}[g] \neq \bot$ and $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$(g.\mathsf{gpk}, g.\mathsf{epk}) \leftarrow g$
$m_{\mathsf{DS}} \leftarrow m \,\|\, vk_u \,\|\, g.\mathsf{gpk}$
$s \leftarrow \mathsf{DS.Sig}(sk_u, m_{\mathsf{DS}})$
$m_{\mathsf{NE}} \leftarrow m \,\|\, vk_u \,\|\, s$
$c \leftarrow \mathsf{NE.Enc}(K_g, n, m_{\mathsf{NE}})$
$S \leftarrow S \cup \{(u, m_{\mathsf{DS}}, s)\}$
$C \leftarrow C \cup \{((g, u, n, m, ad), c)\}$; Return c

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 IUF game as per Fig. 4.

VerDec(g, u, n, c, ad) // $n = \varepsilon$

require $\mathsf{K}[g] \neq \bot$ and $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$(g.\mathsf{gpk}, g.\mathsf{epk}) \leftarrow g$
$m_{\mathsf{NE}} \leftarrow \mathsf{NE.Dec}(K_g, n, c)$
If $m_{\mathsf{NE}} = \bot$ then return $\bot$
$m \,\|\, vk_u \,\|\, s \leftarrow m_{\mathsf{NE}}$
$m_{\mathsf{DS}} \leftarrow m \,\|\, vk_u \,\|\, g.\mathsf{gpk}$
If $\neg\mathsf{DS.Ver}(vk_u, m_{\mathsf{DS}}, s)$ then return $\bot$
$z \leftarrow ((g, u, n, m, ad), c)$
If $\exists g', n', c' : ((g, u, n, m, ad), c) \in C$ then return m
If $\neg\mathsf{user_is_corrupt}[u]$ then
  If $(u, c_{\mathsf{DS}}, s) \notin S$ then
    $\mathsf{bad}_0 \leftarrow \mathsf{true}$ // $\mathcal{A}_{\mathsf{SUFCMA}}$ breaks SUFCMA of DS.
    win $\leftarrow$ true // $\mathcal{G}_{[0,1)}$
  Else // $(u, c_{\mathsf{DS}}, s) \in S$
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$ // This step is unreachable
    win $\leftarrow$ true // $\mathcal{G}_{[0,2)}$
Return m

Adversary $\mathcal{A}_{\mathsf{SUFCMA}}^{\mathrm{U,Sign}}$

$\mathcal{A}_{\mathsf{IUF}}^{\mathrm{SimU,SimG,SimSigEnc,SimVerDec}}$

SimSigEnc(g, u, n, m, ad)

require $\mathsf{K}[g] \neq \bot$ and $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$(g.\mathsf{gpk}, g.\mathsf{epk}) \leftarrow g$
$m_{\mathsf{DS}} \leftarrow m \,\|\, vk_u \,\|\, g.\mathsf{gpk}$
$s \leftarrow$ Sign(u, m_{DS})
$m_{\mathsf{NE}} \leftarrow m \,\|\, vk_u \,\|\, s$
$c \leftarrow \mathsf{NE.Enc}(K_g, n, m_{\mathsf{NE}})$
$S \leftarrow S \cup \{(u, m_{\mathsf{DS}}, s)\}$
$C \leftarrow C \cup \{((g, u, n, m, ad), c)\}$; Return c

SimNewHonGroup(g, users)
SimExposeGroup(g)
SimNewCorrGroup(g, K, users)
// These oracles are identical to the corresponding
// oracles in the IUF game for StE as per Fig. 4.

SimVerDec(g, u, n, c, ad)

require $\mathsf{K}[g] \neq \bot$ and $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$(g.\mathsf{gpk}, g.\mathsf{epk}) \leftarrow g$
$m_{\mathsf{NE}} \leftarrow \mathsf{NE.Dec}(K_g, n, c)$
If $m_{\mathsf{NE}} = \bot$ then return $\bot$
$m \,\|\, vk_u \,\|\, s \leftarrow m_{\mathsf{NE}}$
$m_{\mathsf{DS}} \leftarrow m \,\|\, vk_u \,\|\, g.\mathsf{gpk}$
If $\neg\mathsf{DS.Ver}(vk_u, m_{\mathsf{DS}}, s)$ then return $\bot$
$z \leftarrow ((g, u, n, m, ad), c)$
If $\exists g', n', c' : ((g, u, n, m, ad), c) \in C$ then return m
If $\neg\mathsf{user_is_corrupt}[u]$ then
  If $(u, m_{\mathsf{DS}}, s) \notin S$ then
    $z \leftarrow (u, m_{\mathsf{DS}}, s)$
    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 $\mathsf{sk}[u] \neq \bot$
$\mathsf{user_is_corrupt}[u] \leftarrow$ true
$\mathsf{sk}[u] \leftarrow$ ExposeUser(u)
Return $\mathsf{sk}[u]$

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

Fig. 22: **Top pane:** Games $\mathcal{G}_0$–$\mathcal{G}_2$ for the proof of Theorem 3. The code highlighted in gray was added by expanding the symmetric signcryption algorithms SesLG-StE.SigEnc, SesLG-StE.VerDec in game $\mathcal{G}^{\mathsf{IUF}}_{\mathsf{SesLG\text{-}StE},\mathsf{pred}^{\text{euf-except-nonce-and-group}}_{\text{trivial}}}(\mathcal{A}_{\mathsf{IUF}})$. The code added for the transitions between games is highlighted in purple. **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 $\mathcal{G}_0$.

C.2 Out-group Authenticated Encryption of SesLG-StE (Proof of Theorem 4)

OVERVIEW. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_2$ in Fig. 23 to establish the following claims.

1. $\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{SesLG\text{-}StE},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1$
2. $\Pr[\mathcal{G}_0] = \Pr[\mathcal{G}_1]$
3. $\Pr[\mathcal{G}_1] = \Pr[\mathcal{G}_2]$
4. $2 \cdot \Pr[\mathcal{G}_2] - 1 \leq \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}})$

By combining the above, we have

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

The sequence of hybrids we use for this proof mirrors those used to prove the OAE security of MLS-StE in Appendix B. So the justifications for the corresponding claims carry over analogously from those for MLS-StE. We do not provide individual descriptions to justify the our claims. Readers interested in understanding each transition can refer to the relevant explanations in the prior section while considering Fig. 23.

Games $\mathcal{G}_0$–$\mathcal{G}_2$

$b \leftarrow\!\!\$\ \{0,1\}$; $b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{U,G,SigEnc,VerDec}}_{\mathsf{OAE}}$
Return $b = b'$

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 $(g, n) \notin N$
If $m_0 \neq m_1$ then
  If $\mathsf{group_is_corrupt}[g]$ then return $\bot$
  Else $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$(g.\mathsf{gpk}, g.\mathsf{epk}) \leftarrow g$
$m_{\mathsf{DS}} \leftarrow m \,\|\, vk_u \,\|\, g.\mathsf{gpk}$
$s \leftarrow \mathsf{DS.Sig}(sk_u, m_{\mathsf{DS}})$
$m_{\mathsf{NE}} \leftarrow m \,\|\, vk_u \,\|\, s$
$c \leftarrow \mathsf{NE.Enc}(K_g, n, m_{\mathsf{NE}})$
$C \leftarrow C \cup \{((g, u, n, m_b, ad), c)\}$
$N \leftarrow N \cup \{(g, n)\}$
$W[g, u, n, c, ad] \leftarrow m_1$; Return c

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 return $\bot$ // $\mathcal{G}_{[1,\infty)}$
$(g.\mathsf{gpk}, g.\mathsf{epk}) \leftarrow g$
$m_{\mathsf{NE}} \leftarrow \mathsf{NE.Dec}(K_g, n, c)$
If $m_{\mathsf{NE}} = \bot$ then return $\bot$
$m \,\|\, vk_u \,\|\, s \leftarrow m_{\mathsf{NE}}$
$m_{\mathsf{DS}} \leftarrow m \,\|\, vk_u \,\|\, g.\mathsf{gpk}$
If $\neg\mathsf{DS.Ver}(vk_u, m_{\mathsf{DS}}, s)$ then return $\bot$
$z \leftarrow ((g, u, n, m, ad), c)$ // $\mathcal{G}_{[0,2)}$
If $\exists u'$: $((g, u', n, m, ad), c) \in C$ then return $\bot$ // $\mathcal{G}_{[0,2)}$
If $b = 0$ then return $\bot$ 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 SesLG-StE as per Fig. 4.

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

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 $u \in \mathsf{members}[g]$
If $m_0 \neq m_1$ then
  If $\mathsf{group_is_corrupt}[g]$ then return $\bot$
  $\mathsf{chal}[g] \leftarrow \mathsf{true}$
  $\forall d \in \{0,1\}, m_{\mathsf{Sig},d} \leftarrow m \,\|\, vk_u \,\|\, g.\mathsf{gpk}$
  $\forall d \in \{0,1\}, s_d \leftarrow\!\!\$\ \mathsf{DS.Sig}(\mathsf{sk}[u], m_{\mathsf{Sig},d})$
  $m_{\mathsf{NE},d} \leftarrow m_d \,\|\, s_d$
  $ad_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, ad\rangle$
  $c \leftarrow \mathrm{Enc}(\mathsf{K}[g], n, m_{\mathsf{NE},0}, m_{\mathsf{NE},1} ad_{\mathsf{NE}})$
Else // $m_0 = m_1$
  $m_s \leftarrow \langle g, n, m_0, ad\rangle$; $s \leftarrow\!\!\$\ \mathsf{DS.Sig}(\mathsf{sk}[u], m_s)$
  $m_{\mathsf{NE}} \leftarrow m_0 \,\|\, s$; $ad_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, g.\mathsf{ep}, ad\rangle$
  $c \leftarrow \mathrm{Enc}((g, u), n, m_{\mathsf{NE}}, m_{\mathsf{NE}}, ad_{\mathsf{NE}})$
$W[g, u, n, c, ad] \leftarrow m_1$; 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 $\exists u'$: $W[g, u', n, c, ad] \neq \bot$ then return $\bot$
$m_{\mathsf{NE}} \leftarrow \mathsf{NE.Dec}(K_g, n, c)$
If $m_{\mathsf{NE}} = \bot$ then return $\bot$
abort(1) // $m_{\mathsf{NE}} \neq \bot$ possible only when $b = 1$.

SimNewHonUser(u)
SimExposeUser(u)
SimNewCorrUser(u, sk, vk)
// These oracles are identical to the corresponding
// oracles in the OAE game for SesLG-StE as per Fig. 4.

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

Fig. 23: **Top pane:** Games $\mathcal{G}_0$–$\mathcal{G}_2$ for the proof of Theorem 4. The code highlighted in gray was added by expanding the symmetric signcryption algorithms SesLG-StE.SigEnc, SesLG-StE.VerDec and the ciphertext-triviality predicate $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ in game $\mathcal{G}^{\mathsf{OAE}}_{\mathsf{SesLG\text{-}StE},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}})$. The code added for the transitions between games is highlighted in purple. **Bottom pane:** Adversary $\mathcal{A}_{\mathsf{AEAD}}$ for the proof of Theorem 4. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_2$.

D Proofs for Signal

D.1 In-group UnForgeability of Sig-EtS (Proof of Theorem 5)

OVERVIEW. We use games $\mathcal{G}_0$ through $\mathcal{G}_2$ in Fig. 24 to establish the following claims.

1. $\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{Sig\text{-}EtS},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) = \Pr[\mathcal{G}_0]$
2. $\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$
3. $\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \leq \Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$
4. $\Pr[\mathcal{G}_2] = 0$
5. $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})$
6. $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}] = 0$

By combining the above, we have

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

The sequence of hybrids we use for this proof mirrors those used to prove the IUF security of MLS-StE in Appendix B, except the ciphertext triviality predicate $\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}$ used in this proof is different from $\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}$ used for MLS-StE. Still, the justifications for the corresponding claims follow analogously from those for MLS-StE. The only difference is the justification for the bound on $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$ (claim 6). We justify this claim below. We do not provide individual descriptions to justify the other claims. Readers interested in understanding each transition can refer to the relevant explanations in the prior section while considering Fig. 24.

BOUNDING $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$. The flag bad_1 is only set in game $\mathcal{G}_1$ when $((g, u, \varepsilon, m, ad), c) \notin C$ and $(u, c_{\mathsf{DS}}, s) \in S$ are simultaneously true. We can use the definition c_{DS} to expand $(u, c_{\mathsf{DS}}, s) \in S$ into $(u, \langle ad, g, c_{\mathsf{NE}}\rangle, s) \in S$ where $c = (c_{\mathsf{NE}}, s)$ Therefore, bad_1 is only set in $\mathcal{G}_1$ when $((g, u, n, m, ad), c) \notin C$ and $(u, \langle ad, g, c_{\mathsf{NE}}\rangle, s) \in S$. The latter means that there was a previous call to SIGENC of the form $(g, u, \tilde{n}, \tilde{m}, ad)$ such that $\tilde{m} = \mathsf{NE.Dec}(\mathsf{K}[g], \tilde{n}, c_{\mathsf{NE}})$. However, we know that for a given group g, the adversary queries the same nonce n to SIGENC and VERDEC. Therefore, $\tilde{n} = n$. By the decryption correctness of NE, we have $\tilde{m} = m$. Then by definition of the SIGENC oracle we have that $((g, u, n, m, ad), (c_{\mathsf{NE}}, s)) \in C$. But the first condition gives us $((g, u, n, m, ad), c) \notin C$. We get a contradiction, and hence

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

Games $\mathcal{G}_0$–$\mathcal{G}_2$

$\mathcal{A}_{\mathsf{IUF}}^{\mathrm{U,G,SigEnc,VerDec}}$; Return win

SigEnc(g, u, n, m, ad)

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

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 IUF game as per Fig. 4.

VerDec(g, u, n, c, ad)

require $\mathsf{K}[g] \neq \bot$ and $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$(c_{\mathsf{NE}}, s) \leftarrow c$; $c_{\mathsf{DS}} \leftarrow \langle ad, g, c_{\mathsf{NE}} \rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], c_{\mathsf{DS}}, s)$ then return $\bot$
$m \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], c_{\mathsf{NE}})$
$z \leftarrow ((g, u, n, m, ad), c)$
If $(g, u, n, m, ad), c) \in C$ then return m
If $\neg\mathsf{user_is_corrupt}[u]$ then
  If $(u, c_{\mathsf{DS}}, s) \notin S$ then
    $\mathsf{bad}_0 \leftarrow \mathsf{true}$ // $\mathcal{A}_{\mathsf{SUFCMA}}$ breaks SUFCMA of DS.
    win $\leftarrow$ true // $\mathcal{G}_{[0,1)}$
  Else // $(u, c_{\mathsf{DS}}, s) \in S$
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$ // This step is unreachable
    win $\leftarrow$ true // $\mathcal{G}_{[0,2)}$
Return m

Adversary $\mathcal{A}_{\mathsf{SUFCMA}}^{\mathrm{U,Sign}}$

$\mathcal{A}_{\mathsf{IUF}}^{\mathrm{SimU,SimG,SimSigEnc,SimVerDec}}$

SimSigEnc(g, u, n, m, ad)

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

SimNewHonGroup(g, users)
SimExposeGroup(g)
SimNewCorrGroup(g, K, users)
// These oracles are identical to the corresponding
// oracles in the IUF game for StE as per Fig. 4.

SimVerDec(g, u, n, c, ad)

require $\mathsf{K}[g] \neq \bot$ and $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
require $g.\mathsf{userId} = u$
$(c_{\mathsf{NE}}, s) \leftarrow c$; $c_{\mathsf{DS}} \leftarrow \langle ad, g, c_{\mathsf{NE}} \rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], c_{\mathsf{DS}}, s)$ then return $\bot$
$m \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], c_{\mathsf{NE}})$
$z \leftarrow ((g, u, n, m, ad), c)$
If $(g, u, n, m, ad), c) \in C$ then return m
If $\neg\mathsf{user_is_corrupt}[u]$ then
  If $(u, c_{\mathsf{DS}}, s) \notin S$ then
    $z \leftarrow (u, c_{\mathsf{DS}}, s)$
    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. 24: **Top pane:** Games $\mathcal{G}_0$–$\mathcal{G}_2$ for the proof of Theorem 5. The code highlighted in gray was added by expanding the symmetric signcryption algorithms Sig-EtS.SigEnc, Sig-EtS.VerDec in game $\mathcal{G}^{\mathsf{IUF}}_{\mathsf{Sig\text{-}EtS},\mathsf{pred}^{\mathsf{suf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}})$. The code added for the transitions between games is highlighted in purple. **Bottom pane:** Adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ for the proof of Theorem 5. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_0$.

Games $\mathcal{G}_0$–$\mathcal{G}_4$

$b \leftarrow\!\!\$\ \{0,1\}$; $b' \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{OAE}}^{\mathrm{U,G,SigEnc,VerDec}}$
Return $b = b'$

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 $u \in \mathsf{members}[g]$
If $m_0 \neq m_1$ then
  If $\mathsf{group_is_corrupt}[g]$ then return $\bot$
  $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$c_{\mathsf{NE}} \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m)$
$c_{\mathsf{DS}} \leftarrow \langle ad, g, c_{\mathsf{NE}} \rangle$
$s \leftarrow\!\!\$\ \mathsf{DS.Sig}(\mathsf{sk}[u], c_{\mathsf{DS}})$
$C \leftarrow C \cup \{((g, u, n, m_b, ad), c)\}$
$S \cup \{(u, c_{\mathsf{DS}}, s)\}$
Return (c_{NE}, s)

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. 4.

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 $u \in \mathsf{members}[g]$
$(c_{\mathsf{NE}}, s) \leftarrow c$
$c_{\mathsf{DS}} \leftarrow \langle ad, g, c_{\mathsf{NE}} \rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], c_{\mathsf{DS}}, s)$ then return $\bot$
$m \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], c_{\mathsf{NE}})$
$z \leftarrow ((g, u, n, m, ad), c)$
If $\exists u', c' : ((g, u', n, m, ad), c') \in C$ then return $\bot$
If $b = 0$ then return $\bot$
If $((\mathsf{K}[g], n, m), c_{\mathsf{NE}}) \in C_{\mathsf{NE}}$ then
  If $(u, c_{\mathsf{DS}}, s) \notin S$ then
    $\mathsf{bad}_0 \leftarrow \mathsf{true}$ // $\mathcal{A}_{\mathsf{OAE}}$ broke SUFCMA of DS.
    Return m // $\mathcal{G}_{[0,1)}$
    Return $\bot$ // $\mathcal{G}_{[1,\infty)}$
  Else // $((g, u, n, m, ad), c) \in C$
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$ // Unreachable in $\mathcal{G}_1$
    Return m // $\mathcal{G}_{[0,2)}$
    Return $\bot$ // $\mathcal{G}_{[2,\infty)}$
Else // $((\mathsf{K}[g], n, m), c_{\mathsf{NE}}) \notin C_{\mathsf{NE}}$
  If $(u, c_{\mathsf{DS}}, s) \notin S$ then
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$ // $\mathcal{A}_{\mathsf{OAE}}$ broke SUFCMA of DS.
    Return m // $\mathcal{G}_{[0,3)}$
    Return $\bot$ // $\mathcal{G}_{[3,\infty)}$
  Else // $((g, u, n, m, ad), c) \in C$
    $\mathsf{bad}_3 \leftarrow \mathsf{true}$ // $\mathcal{A}_{\mathsf{OAE}}$ broke correctness of NE.
    Return m // $\mathcal{G}_{[0,4)}$
    Return $\bot$ // $\mathcal{G}_{[4,\infty)}$

Fig. 25: Games $\mathcal{G}_0$–$\mathcal{G}_4$ for the proof of Theorem 6. The code highlighted in gray was added by expanding the symmetric signcryption algorithms Sig-EtS.SigEnc, Sig-EtS.VerDec in game $\mathcal{G}^{\mathsf{OAE}}_{\mathsf{Sig\text{-}EtS},\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}})$. The code added for the transitions between games is highlighted in purple.

D.2 Out-group Authenticated Encryption of Sig-EtS (Proof of Theorem 6)

Overview. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_4$ in Fig. 25 to establish the following claims.

1. $\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{Sig\text{-}EtS},\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1$
2. $\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$
3. $\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \leq \Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$
4. $\Pr[\mathcal{G}_2] - \Pr[\mathcal{G}_3] \leq \Pr[\mathsf{bad}_2^{\mathcal{G}_2}]$
5. $\Pr[\mathcal{G}_3] - \Pr[\mathcal{G}_4] \leq \Pr[\mathsf{bad}_3^{\mathcal{G}_3}]$
6. $2 \cdot \Pr[\mathcal{G}_4] - 1 \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{IND\text{-}CPA}})$
7. $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})$
8. $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}] = 0$
9. $\Pr[\mathsf{bad}_2^{\mathcal{G}_2}] \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})$
10. $\Pr[\mathsf{bad}_3^{\mathcal{G}_3}] = 0$

By combining the above, we have

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{Sig\text{-}EtS},\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}}) &= 2 \cdot \Pr[\mathcal{G}_0] - 1 = 2 \cdot \left(\sum_{i=0}^{3} (\Pr[\mathcal{G}_i] - \Pr[\mathcal{G}_{i+1}]) + \Pr[\mathcal{G}_4] \right) - 1 \\
&= 2 \cdot \left(\sum_{i=0}^{3} \Pr[\mathsf{bad}_i^{\mathcal{G}_i}] + \Pr[\mathcal{G}_4] \right) - 1 \\
&=\leq 2 \cdot \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}}) + \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}}).
\end{aligned}$$

We justify our claims in the following paragraphs.

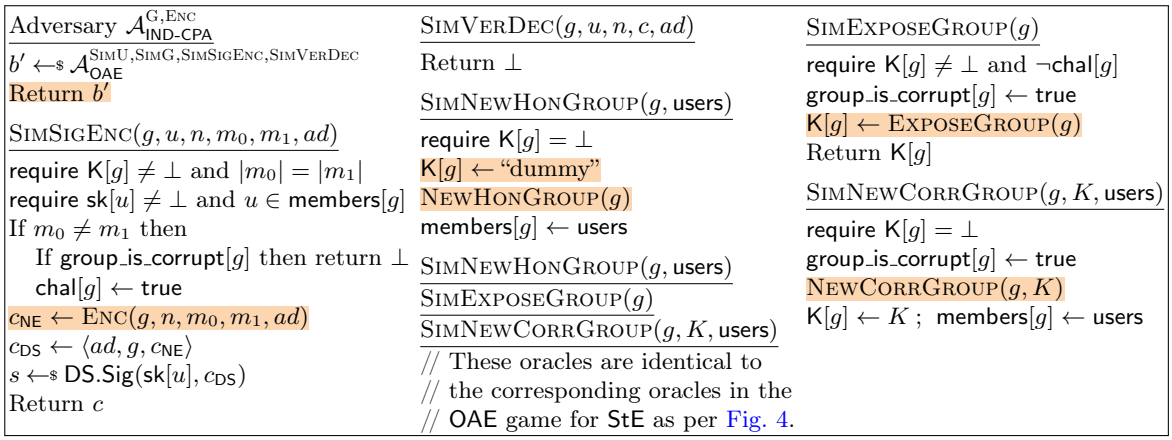


Fig. 26: Adversary $\mathcal{A}_{\mathsf{AEAD}}$ for the proof of Theorem 6. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_2$.

Game $\mathcal{G}_0$. Game $\mathcal{G}_0$ is functionally equivalent to game $\mathcal{G}^{\mathsf{OAE}}_{\mathsf{Sig\text{-}EtS},\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}})$. 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 purple and does not affect its functionality; this code will be used for transitions between games. In particular, the `if-then-else` statements that were added in the VerDec oracle return the decrypted message m in each of the conditional branches. It follows that

$$\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{Sig\text{-}EtS},\mathsf{pred}^{\mathsf{euf}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1.$$

Transitions from $\mathcal{G}_0$ to $\mathcal{G}_4$. Let $i \in \{3\}$. 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 returns the message m. In the game $\mathcal{G}_4$, the VerDec oracle always returns $\bot$.

Bounding $\Pr[\mathcal{G}_4]$. In Fig. 26 we build an adversary $\mathcal{A}_{\mathsf{IND\text{-}CPA}}$ against the IND-CPA security of NE that simulates the game $\mathcal{G}_4$ for adversary $\mathcal{A}_{\mathsf{OAE}}$. Adversary $\mathcal{A}_{\mathsf{IND\text{-}CPA}}$ simulates the group oracles for $\mathcal{A}_{\mathsf{OAE}}$ by calling its own group oracles, and it simulates the user oracles by sampling and maintaining its own keys for DS. Adversary $\mathcal{A}_{\mathsf{IND\text{-}CPA}}$ simulates the SigEnc oracle for $\mathcal{A}_{\mathsf{OAE}}$ by evaluating NE.Enc using its encryption oracle Enc, the VerDec oracle by always returning $\bot$. Finally, when $\mathcal{A}_{\mathsf{OAE}}$ returns a bit b', $\mathcal{A}_{\mathsf{IND\text{-}CPA}}$ returns b' to its own game. It follows that

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{IND\text{-}CPA}}) \geq 2 \cdot \Pr[\mathcal{G}_4] - 1.$$

Bounding $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$. In Fig. 27 we build an adversary $\mathcal{A}^1_{\mathsf{SUFCMA}}$ against the SUFCMA security of DS that simulates the game $\mathcal{G}_0$ for adversary $\mathcal{A}_{\mathsf{OAE}}$. Adversary $\mathcal{A}^1_{\mathsf{SUFCMA}}$ simulates the user oracles for $\mathcal{A}_{\mathsf{OAE}}$ by calling its own user oracles, and it simulates the group oracles by sampling and maintaining its own keys for NE. Adversary $\mathcal{A}^1_{\mathsf{SUFCMA}}$ simulates the SigEnc oracle for $\mathcal{A}_{\mathsf{OAE}}$ by evaluating DS.Sig using its signing oracle Sign and the VerDec oracle by executing all the lines of code locally.

Now consider that conditions that are required to be met for bad_0 to be set in $\mathcal{G}_0$. These are / $\exists u', c' : ((g, u', n, m, ad), c') \in C$, $\mathsf{DS.Ver}(\mathsf{vk}[u], c_{\mathsf{DS}}, s)$, and $(u, c_{\mathsf{DS}}, s) \notin S$. Additionally, $\neg\mathsf{user_is_corrupt}[u]$ always holds as the adversary $\mathcal{A}_{\mathsf{OAE}}$ never queries its ExposeUser or NewCorrUser oracles. Therefore, whenever the bad_0 flag is set in the simulated $\mathcal{G}_0$, $\neg\mathsf{user_is_corrupt}[u]$, $\mathsf{DS.Ver}(\mathsf{vk}[u], c_{\mathsf{DS}}, s)$, and $(u, c_{\mathsf{DS}}, s) \notin S$ simultaneously hold. These are the winning conditions for $\mathcal{A}^1_{\mathsf{SUFCMA}}$. So whenever bad_0 is set in the simulated game $\mathcal{G}_0$, it halts with **abort**(u, c_{DS}, s). We have

$$\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}^1_{\mathsf{SUFCMA}}).$$

Adversaries $\mathcal{A}^1_{\mathsf{SUFCMA}}$ and $\mathcal{A}^2_{\mathsf{SUFCMA}}$

Run $\mathcal{A}_{\mathsf{OAE}}^{\textsc{SimU},\textsc{SimG},\textsc{SimSigEnc},\textsc{SimVerDec}}$

$\textsc{SimSigEnc}(g, u, n, m_0, m_1, ad)$

require $\mathsf{K}[g] \neq \bot$ and $|m_0| = |m_1|$
require $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
If $m_0 \neq m_1$ then
  If $\mathsf{group_is_corrupt}[g]$ then return $\bot$
  $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$c_{\mathsf{NE}} \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m)$; $c_{\mathsf{DS}} \leftarrow \langle ad, g, c_{\mathsf{NE}} \rangle$
$s \leftarrow\!\!\$\ \textsc{Sign}(u, c_{\mathsf{DS}})$
$C \leftarrow C \cup \{((g, u, n, m_b, ad), c)\}$; $S \cup \{(u, c_{\mathsf{DS}}, s)\}$
Return (c_{NE}, s)

$\textsc{SimNewHonGroup}(g, \mathsf{users})$

$\textsc{SimExposeGroup}(g)$

$\textsc{SimNewCorrGroup}(g, K, \mathsf{users})$

// These oracles are identical to the corresponding
// oracles in the IUF game for StE as per Fig. 4.

$\textsc{SimVerDec}(g, u, n, c, ad)$

require $\mathsf{K}[g] \neq \bot$ and $\neg\mathsf{group_is_corrupt}[g]$
require $\mathsf{vk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
$(c_{\mathsf{NE}}, s) \leftarrow c$; $c_{\mathsf{DS}} \leftarrow \langle ad, g, c_{\mathsf{NE}} \rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], c_{\mathsf{DS}}, s)$ then return $\bot$
$m \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], c_{\mathsf{NE}})$
$z \leftarrow ((g, u, n, m, ad), c)$
If $\exists u', c' \colon ((g, u', n, m, ad), c') \in C$ then return $\bot$
If $b = 0$ then return $\bot$
If $((\mathsf{K}[g], n, m), c_{\mathsf{NE}}) \in C_{\mathsf{NE}}$ then
  If $(u, c_{\mathsf{DS}}, s) \notin S$ then return m
  Else
    $z \leftarrow (u, c_{\mathsf{DS}}, s)$ // $\mathcal{A}^1_{\mathsf{SUFCMA}}$
    abort(z) // $\mathcal{A}^1_{\mathsf{SUFCMA}}$
Else
  If $(u, c_{\mathsf{DS}}, s) \notin S$ then return m
  Else
    $z \leftarrow (u, c_{\mathsf{DS}}, s)$ // $\mathcal{A}^2_{\mathsf{SUFCMA}}$
    abort(z) // $\mathcal{A}^2_{\mathsf{SUFCMA}}$

$\textsc{SimNewHonUser}(u)$

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. 27: Adversaries $\mathcal{A}^1_{\mathsf{SUFCMA}}$ and $\mathcal{A}^2_{\mathsf{SUFCMA}}$ for the proof of Theorem 6. The highlighted instructions mark the changes in the code of the simulated games $\mathcal{G}_0$ and $\mathcal{G}_2$ respectively. We use comments to highlight the lines executed by a particular adversary.

Bounding $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$. Let us consider the conditions that need to me met for bad_1 to be set in $\mathcal{G}_1$. These are $\nexists u', c' \colon ((g, u', n, m, ad), c') \in C$, $((\mathsf{K}[g], n, m), c_{\mathsf{NE}}) \in C_{\mathsf{NE}}$, and $(u, c_{\mathsf{DS}}, s) \in S$. We can use the definition of c_{DS} to expand the last one into $(u, \langle ad, g, c_{\mathsf{NE}} \rangle, s) \in S$. Combining this with $((\mathsf{K}[g], n, m), c_{\mathsf{NE}}) \in C_{\mathsf{NE}}$, we know that there was a previous query to SigEnc of the form (g, u, n, m, ad) that produced the ciphertext (c_{NE}, s). By the definition of SigEnc, we have $((g, u, n, m, ad), (c_{\mathsf{NE}}, s)) \in C$. However, from the first condition we have that $\nexists u', c' \colon ((g, u', n, m, ad), c') \in C$. We have a contradiction and hence

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

Bounding $\Pr[\mathsf{bad}_2^{\mathcal{G}_2}]$. In Fig. 27 we build an adversary $\mathcal{A}^2_{\mathsf{SUFCMA}}$ against the SUFCMA security of DS that simulates the game $\mathcal{G}_0$ for adversary $\mathcal{A}_{\mathsf{OAE}}$. Adversary $\mathcal{A}^2_{\mathsf{SUFCMA}}$ simulates the user oracles for $\mathcal{A}_{\mathsf{OAE}}$ by calling its own user oracles, and it simulates the group oracles by sampling and maintaining its own keys for NE.

Adversary $\mathcal{A}^2_{\mathsf{SUFCMA}}$ simulates the SigEnc oracle for $\mathcal{A}_{\mathsf{OAE}}$ by evaluating DS.Sig using its signing oracle Sign. Adversary $\mathcal{A}^2_{\mathsf{SUFCMA}}$ simulates the VerDec oracle for by executing all the lines of code locally.

Now consider that conditions that are required to be met for bad_0 to be set in $\mathcal{G}_0$. These are $\nexists u', c' \colon ((g, u', n, m, ad), c') \in C$, $\mathsf{DS.Ver}(\mathsf{vk}[u], c_{\mathsf{DS}}, s)$, and $(u, c_{\mathsf{DS}}, s) \notin S$. Additionally, $\neg\mathsf{user_is_corrupt}[u]$ always holds as the adversary $\mathcal{A}_{\mathsf{OAE}}$ never queries its ExposeUser or NewCorrUser oracles. Therefore, whenever the bad_0 flag is set in the simulated $\mathcal{G}_0$, $\neg\mathsf{user_is_corrupt}[u]$, $\mathsf{DS.Ver}(\mathsf{vk}[u], c_{\mathsf{DS}}, s)$, and $(u, c_{\mathsf{DS}}, s) \notin S$ simultaneously hold. These are the winning conditions for $\mathcal{A}^2_{\mathsf{SUFCMA}}$. So whenever bad_0 is set in the simulated game $\mathcal{G}_0$, it halts with **abort**(u, c_{DS}, s). We have

$$\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}^2_{\mathsf{SUFCMA}}).$$

BOUNDING $\Pr[\mathsf{bad}_3^{\mathcal{G}_3}]$. Let us consider the conditions that need to me met for bad_3 to be set in $\mathcal{G}_3$. These are $\nexists u', c' \colon ((g, u', n, m, ad), c') \in C$, $((\mathsf{K}[g], n, m), c_{\mathsf{NE}}) \notin C_{\mathsf{NE}}$, and $(u, c_{\mathsf{DS}}, s) \in S$. We can use the definition of c_{DS} to expand the last one into $(u, \langle ad, g, c_{\mathsf{NE}} \rangle, s) \in S$. So there must have been a previous query to SIGENC of the form $(g, u, n, \tilde{m}, ad)$ that produced the ciphertext (c_{NE}, s). However, the decryption correctness of NE we have that $\tilde{m} = m$. This means that the previous query to SIGENC was of the form (g, u, n, m, ad). By the definition of SIGENC, we have that $((g, u, n, m, ad), (c_{\mathsf{NE}}, s))$. But the first condition gives $\nexists u', c' \colon ((g, u', n, m, ad), c') \in C$. We have a contradiction and hence

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

E Proofs for Matrix

E.1 In-group UnForgeability of Mat-EtS (Proof of Theorem 7)

OVERVIEW. We use games $\mathcal{G}_0$ through $\mathcal{G}_2$ in Fig. 28 to establish the following claims.

1. $\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{Mat\text{-}EtS},\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) = \Pr[\mathcal{G}_0]$
2. $\Pr[\mathcal{G}_0] - \Pr[\mathcal{G}_1] \leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}]$
3. $\Pr[\mathcal{G}_1] - \Pr[\mathcal{G}_2] \leq \Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$
4. $\Pr[\mathcal{G}_2] = 0$
5. $\Pr[\mathsf{bad}_0^{\mathcal{G}_0}] \leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}})$
6. $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}] = 0$

By combining the above, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IUF}}_{\mathsf{Mat\text{-}EtS},\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{IUF}}) = \Pr[\mathcal{G}_0] &= \sum_{i=0}^{1}(\Pr[\mathcal{G}_i] - \Pr[\mathcal{G}_{i+1}]) + \Pr[\mathcal{G}_2] \\ &\leq \Pr[\mathsf{bad}_0^{\mathcal{G}_0}] + \Pr[\mathsf{bad}_1^{\mathcal{G}_1}] \\ &\leq \mathsf{Adv}^{\mathsf{SUFCMA}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{SUFCMA}}).\end{aligned}$$

The sequence of hybrids we use for this proof mirrors those used to prove the IUF security of MLS-StE in Appendix B, except the ciphertext triviality predicate $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ we use in this proof is different from $\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}nonce\text{-}and\text{-}group}}_{\mathsf{trivial}}$ used for MLS-StE. Still the justifications for the corresponding claims carry over analogously from those for MLS-StE. The only difference is the justification for the bound on $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$ (claim 6). We justify this claim below. We do not provide individual descriptions to justify the other claims. Readers interested in understanding each transition can refer to the relevant explanations in the prior section while considering Fig. 28.

BOUNDING $\Pr[\mathsf{bad}_1^{\mathcal{G}_1}]$. The flag bad_1 is only set in game $\mathcal{G}_1$ when $\nexists g' \colon ((g', u, n, m, ad), c) \in C$ and $(u, c_{\mathsf{DS}}, s) \in S$ are simultaneously true. We can use the definition c_{DS} to expand $(u, c_{\mathsf{DS}}, s) \in S$ into $(u, \langle ad_{\mathsf{NE}}, c_{\mathsf{NE}}\rangle, s) \in S$. Therefore, bad_1 is only set in $\mathcal{G}_1$ when $\nexists g', n', c' \colon ((g', u, n', m, \varepsilon), c') \in C$ and $(u, (m \,\|\, vk_u \,\|\, g.\mathsf{gpk}), s) \in S$. The latter means that there was a previous call to SIGENC of the form $(\tilde{g}, u, \tilde{n}, m, \varepsilon)$ such that $\tilde{g}.\mathsf{gpk} = g.\mathsf{gpk}$. Let $\tilde{c}$ be the ciphertext that was returned by this SIGENC call. Then by definition of the SIGENC oracle we have that $((\tilde{g}, u, \tilde{n}, m, \varepsilon), \tilde{c}) \in C$. But the first condition gives us $\nexists g', n', c' \colon ((g', u, n', m, \varepsilon), c') \in C$. We get a contradiction, and hence

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

Games $\mathcal{G}_0$–$\mathcal{G}_2$

$\mathcal{A}_{\mathsf{IUF}}^{\mathrm{U,G,SigEnc,VerDec}}$; Return win

SigEnc(g, u, n, m, ad)

require $\mathsf{K}[g] \neq \perp$ and $\mathsf{sk}[u] \neq \perp$ and $u \in \mathsf{members}[g]$
$(g.\mathsf{Id}, g.\mathsf{ipk}, g.\mathsf{uvk}, g.\mathsf{iteration}) \leftarrow g$
$m_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, m\rangle$; $ad_{\mathsf{NE}} \leftarrow \langle ad, g.\mathsf{iteration}\rangle$
$c_{\mathsf{NE}} \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_{\mathsf{NE}}, ad_{\mathsf{NE}})$
$c_{\mathsf{DS}} \leftarrow \langle ad_{\mathsf{NE}}, c_{\mathsf{NE}}\rangle$
$s \leftarrow\!\!\$\ \mathsf{DS.Sig}(\mathsf{sk}[u], c_{\mathsf{DS}})$; $c \leftarrow (c_{\mathsf{NE}}, s)$
$S \leftarrow S \cup \{(u, c_{\mathsf{DS}}, s)\}$
$C \leftarrow C \cup \{((g, u, n, m, ad), c)\}$; Return c

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 IUF game as per Fig. 4.

VerDec(g, u, n, c, ad)

require $\mathsf{K}[g] \neq \perp$ and $\mathsf{vk}[u] \neq \perp$ and $u \in \mathsf{members}[g]$
$(g.\mathsf{Id}, g.\mathsf{ipk}, g.\mathsf{uvk}, g.\mathsf{iteration}) \leftarrow g$
$(c_{\mathsf{NE}}, s) \leftarrow c$; $ad_{\mathsf{NE}} \leftarrow \langle ad, g.\mathsf{iteration}\rangle$
$c_{\mathsf{DS}} \leftarrow \langle ad_{\mathsf{NE}}, c_{\mathsf{NE}}\rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], c_{\mathsf{DS}}, s)$ then return $\perp$
$m_{\mathsf{NE}} \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c_{\mathsf{NE}}, ad_{\mathsf{NE}})$
If $m_{\mathsf{NE}} = \perp$then reutrn $\perp$
$\langle G, m\rangle \leftarrow m_{\mathsf{NE}}$
If $G \neq g.\mathsf{Id}$ then return $\perp$
$Z \leftarrow ((g, u, n, m, ad), c)$
If $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}group}}(z, C)$ then return m
If $\neg\mathsf{user_is_corrupt}[u]$ then
  If $(u, c_{\mathsf{DS}}, s) \notin S$ then
    $\mathsf{bad}_0 \leftarrow \mathsf{true}$ // $\mathcal{A}_{\mathsf{SUFCMA}}$ breaks SUFCMA of DS.
    win $\leftarrow$ true // $\mathcal{G}_{[0,1)}$
  Else // $(u, c_{\mathsf{DS}}, s) \in S$
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$ // This step is unreachable
    win $\leftarrow$ true // $\mathcal{G}_{[0,2)}$
Return m

Adversary $\mathcal{A}_{\mathsf{SUFCMA}}^{\mathrm{U,Sign}}$

$\mathcal{A}_{\mathsf{IUF}}^{\mathrm{SimU,SimG,SimSigEnc,SimVerDec}}$

SimSigEnc(g, u, n, m, ad)

require $\mathsf{K}[g] \neq \perp$ and $\mathsf{sk}[u] \neq \perp$ and $u \in \mathsf{members}[g]$
$(g.\mathsf{Id}, g.\mathsf{ipk}, g.\mathsf{uvk}, g.\mathsf{iteration}) \leftarrow g$
$m_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, m\rangle$; $ad_{\mathsf{NE}} \leftarrow \langle ad, g.\mathsf{iteration}\rangle$
$c_{\mathsf{NE}} \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_{\mathsf{NE}}, ad_{\mathsf{NE}})$
$c_{\mathsf{DS}} \leftarrow \langle ad_{\mathsf{NE}}, c_{\mathsf{NE}}\rangle$
$s \leftarrow \mathrm{Sign}(u, c_{\mathsf{DS}})$; $c \leftarrow (c_{\mathsf{NE}}, s)$
$S \leftarrow S \cup \{(u, c_{\mathsf{DS}}, s)\}$
$C \leftarrow C \cup \{((g, u, n, m, ad), c)\}$; Return c

SimNewHonGroup(g, users)

SimExposeGroup(g)

SimNewCorrGroup(g, K, users)

// These oracles are identical to the corresponding
// oracles in the IUF game for Mat-EtS as per Fig. 4.

SimVerDec(g, u, n, c, ad)

require $\mathsf{K}[g] \neq \perp$ and $\mathsf{vk}[u] \neq \perp$ and $u \in \mathsf{members}[g]$
$(g.\mathsf{Id}, g.\mathsf{ipk}, g.\mathsf{uvk}, g.\mathsf{iteration}) \leftarrow g$
$(c_{\mathsf{NE}}, s) \leftarrow c$; $ad_{\mathsf{NE}} \leftarrow \langle ad, g.\mathsf{iteration}\rangle$
$c_{\mathsf{DS}} \leftarrow \langle ad_{\mathsf{NE}}, c_{\mathsf{NE}}\rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], c_{\mathsf{DS}}, s)$ then return $\perp$
$m_{\mathsf{NE}} \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c_{\mathsf{NE}}, ad_{\mathsf{NE}})$
If $m_{\mathsf{NE}} = \perp$then reutrn $\perp$
$\langle G, m\rangle \leftarrow m_{\mathsf{NE}}$
If $G \neq g.\mathsf{Id}$ then return $\perp$
$z \leftarrow ((g, u, n, m, ad), c)$
If $\neg\mathsf{user_is_corrupt}[u]$ then
  If $(u, c_{\mathsf{DS}}, s) \notin S$ then
    $z \leftarrow (u, c_{\mathsf{DS}}, s)$
    abort(z)
Return m

SimNewHonUser(u)

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

SimExposeUser(u)

require $\mathsf{sk}[u] \neq \perp$
$\mathsf{user_is_corrupt}[u] \leftarrow \mathsf{true}$
$\mathsf{sk}[u] \leftarrow$ ExposeUser(u)
Return $\mathsf{sk}[u]$

SimNewCorrUser(u, sk, vk)

require $\mathsf{sk}[u] = \mathsf{vk}[u] = \perp$
$\mathsf{user_is_corrupt}[u] \leftarrow \mathsf{true}$
NewCorrUser(u, sk, vk)
$\mathsf{sk}[u] \leftarrow sk$; $\mathsf{vk}[u] \leftarrow vk$

Fig. 28: **Top pane:** Games $\mathcal{G}_0$–$\mathcal{G}_2$ for the proof of Theorem 7. The code highlighted in gray was added by expanding the symmetric signcryption algorithms Mat-EtS.SigEnc, Mat-EtS.VerDec in game $\mathcal{G}_{\mathsf{Mat\text{-}EtS},\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}group}}}^{\mathsf{IUF}}(\mathcal{A}_{\mathsf{IUF}})$. The code added for the transitions between games is highlighted in purple. **Bottom pane:** Adversary $\mathcal{A}_{\mathsf{SUFCMA}}$ for the proof of Theorem 7. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_0$.

E.2 Out-group Authenticated Encryption of Mat-EtS (Proof of Theorem 8)

OVERVIEW. This proof uses games $\mathcal{G}_0$ through $\mathcal{G}_2$ in Fig. 29 to establish the following claims.

1. $\mathsf{Adv}^{\mathsf{OAE}}_{\mathsf{Mat\text{-}EtS},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}}) = 2 \cdot \Pr[\mathcal{G}_0] - 1$
2. $\Pr[\mathcal{G}_0] = \Pr[\mathcal{G}_1]$
3. $\Pr[\mathcal{G}_1] = \Pr[\mathcal{G}_2]$
4. $2 \cdot \Pr[\mathcal{G}_2] - 1 \leq \mathsf{Adv}^{\mathsf{AEAD}}_{\mathsf{NE}}(\mathcal{A}_{\mathsf{AEAD}})$

By combining the above, we have

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

The sequence of hybrids we use for this proof mirrors those used to prove the OAE security of MLS-StE in Appendix B, except the ciphertext triviality predicate $\mathsf{pred}^{\mathsf{euf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ is weaker than $\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}$ that was used for MLS-StE. The justifications for the corresponding claims follow analogously from those for MLS-StE. We do not provide individual descriptions to justify the our claims. Readers interested in understanding each transition can refer to the relevant explanations in the prior section while considering Fig. 29.

Games $\mathcal{G}_0$–$\mathcal{G}_2$

$b \leftarrow\$ \{0,1\}$; $b' \leftarrow\$ \mathcal{A}_{\mathsf{OAE}}^{\mathrm{U,G,SigEnc,VerDec}}$; Return $b = b'$

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 $(g, n) \notin N$
If $m_0 \neq m_1$ then
  If $\mathsf{group_is_corrupt}[g]$ then return $\bot$
  $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$(g.\mathsf{Id}, g.\mathsf{ipk}, g.\mathsf{uvk}, g.\mathsf{iteration}) \leftarrow g$
$m_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, m\rangle$; $ad_{\mathsf{NE}} \leftarrow \langle ad, g.\mathsf{iteration}\rangle$
$c_{\mathsf{NE}} \leftarrow \mathsf{NE.Enc}(\mathsf{K}[g], n, m_{\mathsf{NE}}, ad_{\mathsf{NE}})$
$c_{\mathsf{DS}} \leftarrow \langle ad_{\mathsf{NE}}, c_{\mathsf{NE}}\rangle$
$s \leftarrow\$ \mathsf{DS.Sig}(\mathsf{sk}[u], c_{\mathsf{DS}})$; $c \leftarrow (c_{\mathsf{NE}}, s)$
$C \leftarrow C \cup \{((g, u, n, m_b, ad), c)\}$
$N \leftarrow N \cup \{(g, n)\}$
$W[g, u, n, c, ad] \leftarrow m_1$; Return c

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 $\exists u', c' : W[g, u', n, c', ad] \neq \bot$ then return $\bot$ // $\mathcal{G}_{[1,\infty)}$
$(g.\mathsf{Id}, g.\mathsf{ipk}, g.\mathsf{uvk}, g.\mathsf{iteration}) \leftarrow g$
$(c_{\mathsf{NE}}, s) \leftarrow c$; $ad_{\mathsf{NE}} \leftarrow \langle ad, g.\mathsf{iteration}\rangle$
$c_{\mathsf{DS}} \leftarrow \langle ad_{\mathsf{NE}}, c_{\mathsf{NE}}\rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], c_{\mathsf{DS}}, s)$ then return $\bot$
$m_{\mathsf{NE}} \leftarrow \mathsf{NE.Dec}(\mathsf{K}[g], n, c_{\mathsf{NE}}, ad_{\mathsf{NE}})$
If $m_{\mathsf{NE}} = \bot$then reutrn $\bot$
$\langle G, m\rangle \leftarrow m_{\mathsf{NE}}$
If $G \neq g.\mathsf{Id}$ then return $\bot$
$z \leftarrow ((g, u, n, m, ad), c)$ // $\mathcal{G}_{[0,2)}$
If $\exists u', c' : ((g, u', n, m, ad), c') \in C$ then return $\bot$ // $\mathcal{G}_{[0,2)}$
If $b = 0$ then return $\bot$ 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 Mat-EtS as per Fig. 4.

Adversary $\mathcal{A}_{\mathsf{NE}}^{\mathrm{G,Enc,Dec}}$

$b' \leftarrow\$ \mathcal{A}_{\mathsf{OAE}}^{\mathrm{SimU,SimG,SimSigEnc,SimVerDec}}$; Return b'

SimSigEnc(g, u, n, m_0, m_1, ad)

require $\mathsf{K}[g] \neq \bot$ and $|m_0| = |m_1|$
require $\mathsf{sk}[u] \neq \bot$ and $u \in \mathsf{members}[g]$
If $m_0 \neq m_1$ then
  If $\mathsf{group_is_corrupt}[g]$ then return $\bot$
  $\mathsf{chal}[g] \leftarrow \mathsf{true}$
$(g.\mathsf{Id}, g.\mathsf{ipk}, g.\mathsf{uvk}, g.\mathsf{iteration}) \leftarrow g$
$m_{\mathsf{NE}} \leftarrow \langle g.\mathsf{Id}, m\rangle$; $ad_{\mathsf{NE}} \leftarrow \langle ad, g.\mathsf{iteration}\rangle$
$c_{\mathsf{NE}} \leftarrow \mathrm{Enc}(g, n, m_{\mathsf{NE}}, ad_{\mathsf{NE}})$
$c_{\mathsf{DS}} \leftarrow \langle ad_{\mathsf{NE}}, c_{\mathsf{NE}}\rangle$
$s \leftarrow\$ \mathsf{DS.Sig}(\mathsf{sk}[u], c_{\mathsf{DS}})$; $c \leftarrow (c_{\mathsf{NE}}, s)$
$W[g, u, n, c, ad] \leftarrow m_1$; 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 $\exists u', c' : W[g, u', n, c', ad] \neq \bot$ then return $\bot$
$(g.\mathsf{Id}, g.\mathsf{ipk}, g.\mathsf{uvk}, g.\mathsf{iteration}) \leftarrow g$
$(c_{\mathsf{NE}}, s) \leftarrow c$; $ad_{\mathsf{NE}} \leftarrow \langle ad, g.\mathsf{iteration}\rangle$
$c_{\mathsf{DS}} \leftarrow \langle ad_{\mathsf{NE}}, c_{\mathsf{NE}}\rangle$
If $\neg\mathsf{DS.Ver}(\mathsf{vk}[u], c_{\mathsf{DS}}, s)$ then return $\bot$
$m_{\mathsf{NE}} \leftarrow \mathrm{Dec}(g, n, c_{\mathsf{NE}}, ad_{\mathsf{NE}})$
If $m_{\mathsf{NE}} = \bot$ then return $\bot$
abort(1) // $m_{\mathsf{NE}} \neq \bot$ possible only when $b = 1$.

SimNewHonUser(u)
SimExposeUser(u)
SimNewCorrUser(u, sk, vk)
// These oracles are identical to the corresponding
// oracles in the OAE game for Mat-EtS as per Fig. 4.

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

Fig. 29: **Top pane:** Games $\mathcal{G}_0$–$\mathcal{G}_2$ for the proof of Theorem 8. The code highlighted in gray was added by expanding the symmetric signcryption algorithms Mat-EtS.SigEnc, Mat-EtS.VerDec and the ciphertext-triviality predicate $\mathsf{pred}_{\mathsf{trivial}}^{\mathsf{suf\text{-}except\text{-}user}}$ in game $\mathcal{G}^{\mathsf{OAE}}_{\mathsf{Mat\text{-}EtS},\mathsf{pred}^{\mathsf{suf\text{-}except\text{-}user}}_{\mathsf{trivial}}}(\mathcal{A}_{\mathsf{OAE}})$. The code added for the transitions between games is highlighted in purple. **Bottom pane:** Adversary $\mathcal{A}_{\mathsf{NE}}$ for the proof of Theorem 8. The highlighted instructions mark the changes in the code of the simulated game $\mathcal{G}_2$.