Orca: Blocklisting in Sender-Anonymous Messaging

Nirvan Tyagi
Cornell University

Julia Len
Cornell University

Ian Miers
University of Maryland

Thomas Ristenpart
Cornell Tech

Abstract. Sender-anonymous end-to-end encrypted messaging allows sending messages to a recipient without revealing the sender's identity to the messaging platform. Signal recently introduced a sender anonymity feature that includes an abuse mitigation mechanism meant to allow the platform to block malicious senders on behalf of a recipient.

We explore the tension between sender anonymity and abuse mitigation. We start by showing limitations of Signal's deployed mechanism, observing that it results in relatively weak anonymity properties and showing a new griefing attack that allows a malicious sender to drain a victim's battery. We therefore design a new protocol, called Orca, that allows recipients to register a privacy-preserving blocklist with the platform. Without learning the sender's identity, the platform can check that the sender is not on the blocklist and that the sender can be identified by the recipient. We construct Orca using a new type of group signature scheme, for which we give formal security notions. Our prototype implementation showcases Orca's practicality.

1 Introduction

End-to-end (E2E) encrypted messaging, now relied upon by billions of people due to products like Signal, WhatsApp, Facebook Messenger, and more, provides strong E2E confidentiality and integrity guarantees [5, 25]: the messaging platform itself cannot read or modify user messages. The E2E encryption protocols used [56] do not, however, attempt to ensure anonymity, so the platform learns the sender and recipient of every message sent over the network. While academic systems [4, 6, 26, 27, 45, 48, 51, 62, 64, 65] have developed protocols that hide the identity of senders and receivers from platforms, they introduce expensive overheads.

A recent suggestion for pragmatic privacy improvements is to aim solely for sender anonymity. Introduced by Signal in a feature called "sealed sender" [52], sender anonymity ensures that the sender's identity is never revealed via messages to the platform, e.g., the sender does not authenticate with an account password or digital signature; messages reveal only the intended recipient. While sealed sender does not hide network-level identifiers such as IP addresses, one can do so by composing it with Tor [29] or an anonymous broadcast [26, 44, 51, 57, 65].

In this work, we explore a key tension in sender-anonymous systems: mitigating abuse by malicious senders. Already E2E encryption makes some kinds of abuse mitigations, such as content-based moderation, more challenging (c.f., [30, 32, 38, 63]). Sender anonymity complicates the setting further because the lack of sender authentication means that the platform cannot block unwanted messages on behalf of a recipient in a conventional way.

To enable platform blocking, Signal's sealed sender has a user distribute an access key to their contacts that senders must show to the platform when sending the user a sender-anonymous message. If a sender cannot provide an access key, the platform drops the message. A user that blocks a sender in their client triggers a rotation of this key and a redistribution to the (remaining) contacts. Future messages from the blocked sender will be dropped by the platform.

We observe two deficiencies with this approach. First, access keys must be distributed over non-sender-anonymous channels, meaning the platform learns the identities of users who can send sender-anonymous messages to a particular recipient. This significantly lowers the anonymity guarantee—in the limit of having only a single contact, there is no anonymity at all.

Second, we show a simple "griefing" attack that works despite the anti-abuse mechanism. By design, the sender is hidden from the platform, and only the recipient can identify the sender of a sender-anonymous message. However, a malicious sender can trivially craft malformed messages that even the recipient will not be able to identify. The recipient's client rejects these messages, but not before processing them. This is particularly problematic for mobile clients as it uses up battery life; we experimentally verify that an attacker can easily drain a target's battery in a short period of time. To make matters worse, neither victim nor platform can identify the attacker, and so the victim will not know who to block.

We design a new abuse mitigation mechanism for privacy-preserving blocklisting in sender-anonymous messaging. Our protocol, called Orca, allows recipients to register a blocklist with the platform. The blocklist is privacy-preserving, meaning it does not reveal the identities of the blocked users. Senders construct messages that are anonymous to the platform, but can be verified by the platform as being attributable to a sender not present on the blocklist. If the sender is on the blocklist or if the message is malformed, then the platform rejects the message; if the message is delivered, the recipient is guaranteed to be able to identify the sender.

Importantly, Orca provides a new non-interactive initialization functionality that allows a user to initiate sender-anonymous messages without having previously communicated with the recipient. This significantly enhances the anonymity guarantees, because it expands the anonymity set to be as large as all registered users of the system.

In summary, our contributions are:

- We build a threat model for sender-anonymous messaging and identify limitations in previous approaches, including a new griefing attack against Signal's sealed sender that we evaluate.
- We construct a new group signature scheme [24] to make up the core of Orca's functionality. The new primitive is tailored to the needs of our setting and supports multiple openers, keyed verification, and local revocation; see Section 4 for details. We provide new security definitions, building upon ones from prior work [9, 15].
- We show an extension of Orca that integrates mechanisms from anonymous credentials [22] to arrange that the relatively expensive group signature scheme is only used periodically when initiating a new conversation. Initialization will generate a batch of one-time-use sender tokens [46, 47], which can be spent to authenticate messages and replenished at very low cost.
- We implement and evaluate Orca, suggesting that it is sufficiently performant to deploy at scale. In particular, once initialized, the token-based extension incurs only 30B additional bandwidth cost per message and only one extra group exponentiation of computation for clients; the platform need only compute a group exponentiation and check the token against a strikelist. The computational cost for the platform is paid during initialization which incurs work on the order of the size of the recipient's blocklist ($\sim$ 200ms for a blocklist of length 100). We find that a medium-provisioned server can comfortably support a deployment of a million users depending on frequency of conversation initialization.

2 Setting: Sender Anonymity for E2EE

This work focuses on sender-anonymous E2E encrypted messaging hosted by a centralized messaging platform. In this section and throughout the body, we will often use Signal as our running example. However, the techniques that we introduce are relevant for any sender-anonymous messaging system in which the platform learns the recipient identity.

2.1 Background: Signal and Sealed Sender

Non-sender-anonymous E2EE messaging. We first briefly outline Signal's non-sender-anonymous protocol. For simplicity we restrict attention to one client per user. A user wishing to send a message first registers an account with the platform using a long-lived identity public key pk_s, retaining the associated secret key sk_s. The user then must contact the platform to obtain the long-lived public key pk_r of their intended recipient. Once this phase is complete, a client can securely send messages via Signal's *double ratchet* protocol [56]. This provides state-of-the-art message confidentiality guarantees even in the event of key compromise [5, 25].

Signal, like most other E2E encrypted messaging platforms, requires users to authenticate their account when sending and receiving messages. Importantly, this allows for abuse prevention because the platform can block malicious senders, and even block senders from talking to a specific recipient. On the other hand, such account authentication, e.g., via public key signature or unique account password, does not provide cryptographic sender anonymity.

Sender anonymity with sealed sender. Sealed sender is Signal's protocol [52] for cryptographic sender anonymity motivated by their desire to minimize the amount of trust their users must place in the platform. We will now walk through a high level summary of how sealed sender works.

Initialization and key exchange. As before, senders must first register a public key pk_s with the platform. The user is issued a short-lived *sender certificate* from the platform, that we denote by $cert$. The certificate contains a digital signature by the platform in order to attest to the validity of the user's identity key. These certificates must be periodically updated, requiring the user to rerun the registration protocol.

To receive sealed messages a recipient must generate their long-lived identity key pair (pk_r, sk_r) as usual, but now additionally generate a 96-bit *access key* that we denote by ak. Both pk_r and ak are registered with the platform. Looking ahead, senders will need to show ak to the platform to send a sealed message. This means that the recipient must distribute ak to whomever they want to grant the ability to send sealed messages. By default, the access key is distributed to all contacts of a user through Signal's original non-sender-anonymous channel. Additionally, users can opt into accepting sealed messages from anyone, including non-contacts. In this case, senders do not need a recipient's access key to send them sealed messages.

Sending a sealed message. The pseudocode for sending and receiving a message via sealed sender is provided in Figure 1. It is designed to work modularly as a layer on top of any non-sender-anonymous E2E encryption protocol. At a high level, the protocol creates two ciphertexts: (1) an identity ciphertext encrypting the sender's long-lived public key pk_s to the recipient, and (2) a content ciphertext encrypting the standard E2E encryption ciphertext along with the sender certificate. The identity ciphertext and content ciphertext cryptographically hide the sender identity even if the underlying E2E encryption ciphertext does not [1].

More specifically, the protocol encrypts the sender identity

[1] Signal's use of the double ratchet algorithm produces ciphertexts that can either include the sender identity in plaintext or include messaging metadata such as counters used for in-order processing that would leak information useful for linking senders.

```
$\mathsf{SealedSender.Send}(m)$
$ct_m \leftarrow \mathsf{ratchet.Enc}(m)$
$(pk_e, sk_e) \leftarrow\!\!\$\ \mathsf{KeyGen}()$
$\mathsf{salt}_1 \leftarrow (pk_r, pk_e)$
$(e_{\mathsf{chain}}, k_e) \leftarrow \mathsf{HKDF}(\mathsf{salt}_1, pk_r^{sk_e})$
$ct_{id} \leftarrow\!\!\$\ \mathsf{AE.Enc}(k_e, pk_s)$
$\mathsf{salt}_2 \leftarrow (e_{\mathsf{chain}}, ct_{id})$
$k \leftarrow \mathsf{HKDF}(\mathsf{salt}_2, pk_r^{sk_s})$
$ct_{ss} \leftarrow\!\!\$\ \mathsf{AE.Enc}(k, cert \| ct_m)$
Return $(pk_e, ct_{id}, ct_{ss}), ak$
```

```
$\mathsf{SealedSender.Rcv}(pk_e, ct_{id}, ct_{ss})$
$\mathsf{salt}_1 \leftarrow (pk_r, pk_e)$
$(e_{\mathsf{chain}}, k_e) \leftarrow \mathsf{HKDF}(\mathsf{salt}_1, pk_e^{sk_r})$
$pk_s \leftarrow \mathsf{AE.Dec}(k_e, ct_{id})$
$\mathsf{salt}_2 \leftarrow (e_{\mathsf{chain}}, ct_{id})$
$k \leftarrow \mathsf{HKDF}(\mathsf{salt}_2, pk_s^{sk_r})$
$cert \| ct_m \leftarrow \mathsf{AE.Dec}(k, ct_{ss})$
$b \leftarrow \mathsf{Verify}(pk_s, cert)$
If $b = 0$ then return $\perp$
$m \leftarrow \mathsf{ratchet.Dec}(ct_m)$
Return $m$
```

Figure 1: Pseudocode for Signal's sealed sender feature.

pk_s via a variant of hashed ElGamal [2] to produce the identity ciphertext ct_{id}. In particular, it generates ephemeral key pair (pk_e, sk_e) and makes use of a hash-based key derivation function HKDF and authenticated encryption scheme AE. The sender then encrypts the plaintext m using the original double ratchet algorithm $\mathsf{ratchet.Enc}(m)$. It bundles the resulting ciphertext ct_m and sender certificate $cert$ and encrypts this with a key derived from long-lived identity keys pk_s and pk_r to produce the content ciphertext ct_{ss}. The sender indicates the intended recipient and sends the triple (pk_e, ct_{id}, ct_{ss}) along with the recipient's access key ak to the platform.

Upon receipt of the sender's message, the platform checks that the intended recipient's registered access key matches ak. If this check passes, then the platform forwards the triple (pk_e, ct_{id}, ct_{ss}) to the recipient. The recipient decrypts as shown in Figure 1. Once it recovers $cert$ and ct_m, it verifies the sender as a valid account using the certificate and the recovered identity key pk_s. If the sender's identity is authenticated, then ct_m is decrypted using the double ratchet algorithm.

2.2 Limitations of Sealed Sender

There are limitations to Signal's sealed sender protocol for sender anonymity, which we raise here in the form of three different classes of attacks.

Traffic analysis of sender-anonymous messages. An inherent leakage of the sender-anonymous messaging setting (as opposed to the sender- *and* recipient-anonymous setting) is that the recipient of each message is inherently leaked to the platform. Martiny et al. [53] demonstrate a set of statistical disclosure attacks that use this leakage to infer communicating partners, for example, by searching for users with interleaving messages suggesting a back-and-forth conversation pattern. They provide a modification to Signal's sealed sender that protects against traffic analysis of sender-anonymous messages, which they call "sender-anonymous conversations". This mitigation approach, as well as another separate approach which instead relies on random message delays and/or noise messages [57], do not provide solutions for blocklisting. The techniques we introduce for supporting blocklists compose well with these traffic analysis mitigations. Given this prior work, we do not explicitly address traffic analysis of sender-anonymous messages beyond considering the anonymity set, as we discuss next.

Traffic analysis of non-sender-anonymous messages. Recall that access keys are distributed through Signal's original non-sender-anonymous channel. While this setup is still encrypted, the platform nevertheless observes with whom the user exchanged non-sender-anonymous messages. Thus, when a sender anonymously authenticates using ak, the set of users that could correspond to the sender (i.e., the *anonymity set* of the sender) is restricted and known to the platform. This means, for example, if a recipient only has a single contact with which they have communicated, there is no sender anonymity at all. Furthermore, if a user rotates their access key to revoke sending access, this resets their anonymity set of senders, as their new access key must be redistributed.

Martiny et al. [53] assume in their threat model that these access keys have already been exchanged between communicating parties. Their attack can therefore be improved by tracking the sender anonymity set of a recipient learned by the platform. Notably, our solution for blocklisting will prevent such improvements.

Griefing attack by evading identification. Sealed sender relies on the sender to self-identify to the recipient: the platform can not check for malformed messages. Instead, the recipient must decrypt and check validity of the sender identity key and certificate, dropping messages that do not verify. This allows for a straightforward griefing attack in which an attacker can spam the recipient with untraceable messages, causing the recipient's device to suffer battery drain and to consume bandwidth, a type of user-mounted DoS attack.

We demonstrate through a proof-of-concept implementation that this griefing attack is effective. Our attack simply modifies pk_e in (pk_e, ct_{id}, ct_{ss}) to a random value pk_f. To the platform this is indistinguishable from a legitimate sealed sender message, but the recipient's decryption will fail when trying to decrypt ct_{id}. The recipient cannot recover any information about the sender. Running experiments on a Google Pixel phone running Android 9, we find that sending just 1 message every 10 seconds causes the battery to drain at an increased rate of $9\times$ over baseline. We provide more extensive measurements of this attack in Appendix A.

Ultimately, there are no satisfying mitigation options available to victims (see last section of Appendix A). If the victim of the attack has opted in to accepting sealed sender messages from non-contacts, the attack can be mounted by anyone. Otherwise the attacker needs the recipient's access key, meaning the attacker must be one of the victim's contacts (or has found some other way to obtain the access key). While this limits who can mount the attack in the default case, it is still problematic: The victim can rotate their access key ak and attempt to redistribute a new ak' to their communicating partners. If the attacker is not able to get access to the new access key, the attack will be stopped by the platform and no messages will

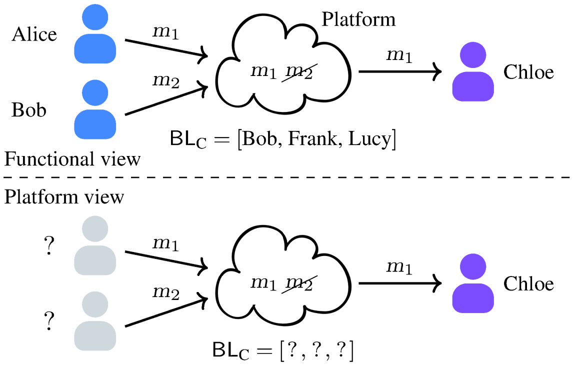
Alice
m_1
Bob
m_2
Platform
m_1 ~~m_2~~
m_1
Chloe
BL_C = [Bob, Frank, Lucy]
Functional view
Platform view
?
?
$\mathsf{BL}_\mathsf{C} = [?, ?, ?]$

Figure 2: Privacy-preserving, outsourced blocklisting for sender-anonymous messaging. The platform is able to block messages from users on Chloe's blocklist without learning their identity. The top view shows the functionality of outsourced blocklisting, while the bottom view shows what is revealed to the platform. Not shown, Chloe can also efficiently identify the sender of message m_1 as Alice and update her blocklist BL_C if needed.

reach the victim's client. But since the attack leaves no information about which of the victim's communicating partners is responsible, the victim can only make a guess as to whom they should block.

Realistically to maintain usability of their mobile device, a user may limit Signal to only a few highly trusted contacts, or will push the user off Signal to a less private messenger. We consider both of these outcomes to be highly damaging to vulnerable users that would benefit from a metadata-private messenger. Looking forward, we will want a mechanism that provides the user more granular recourse against misbehaving senders.

3 Outsourced Blocklisting

We now turn to building a new sender-anonymous messaging protocol that avoids the current weaknesses of sealed sender. Our approach is to enable what we call privacy-preserving outsourced blocklisting (see Figure 2).

Goals. Such a system should enjoy the following features:

- *Sender anonymity:* Messages cryptographically hide the sender identity from the platform.
- *Sender attribution:* Recipients can cryptographically verify the sender of any ciphertexts delivered by the platform.
- *Blocklisting:* Recipients can register a blocklist with the platform and update it efficiently. The platform can use the blocklist to drop sender-anonymous messages from senders that the recipient has added to the blocklist.
- *Blocklist anonymity:* The blocklist should not reveal the identities of the senders blocked by the recipient.

Together these properties prevent the type of griefing attacks that affect sealed sender: a client receiving problematic messages can identify the sender and instruct the platform to drop them on the client's behalf.

We would also like the system to support:

- *Non-interactive initialization:* Users can begin sending sender-anonymous messages without previous interaction with the intended recipient.

This property obviates the use of non-sender-anonymous channels to initiate sender-anonymous communication. In particular, the platform should not be able to attribute messages to some smaller subset of users, as messages can have originated from any registered user of the system.

Orca is designed to accompany a sender-anonymous E2EE messaging protocol to provide the functionality of outsourced blocklisting while carrying over both the sender-anonymity and message confidentiality properties of the underlying protocol. As such, we assume the underlying E2EE protocol is sender-anonymous, and if it is not, can easily be made so using encapsulation techniques similar to sealed sender (see Figure 1). Our protocol will provide a registration process in which users interact with the platform to generate the required keys for the protocol; this will be done at the same time users register for the underlying E2EE protocol. To send a message, the sender first encrypts the message plaintext pt to the recipient as specified by the E2EE protocol. Then, Orca will concern itself with authenticating the delivery of the produced E2EE ciphertext; the authenticity of the underlying message plaintext needs to be provided by the E2EE protocol. We will refer to the E2EE ciphertext as the "message" from Orca's perspective.

Threat model. We assume an active, persistent adversary that controls the messaging platform and an arbitrary number of users. We assume the clients of legitimate users are not compromised and that they correctly abide by the protocol.

Our primary concern is the *cryptographic anonymity* of the messaging protocol. The adversary, even with active deviations from the protocol, should not be able to learn sender identity information from the contents of protocol messages.

Even in the case that anonymity is achieved at the message proctocol layer, identification information can leak through the network layer, e.g., by associating IP addresses or by making inferences based on timing. We consider preventing such leakage to be orthogonal to the goal of providing a blocklisting solution for the message protocol layer: existing solutions for mitigating network leakage will compose. Sender-anonymous channels resilient to linking attacks that exploit IP addresses can be constructed using services such as Tor [29]; linking attacks performed by stronger global network adversaries with the ability to observe and inject traffic along any network link can be mitigated using prior academic solutions for anonymous broadcasting [26, 44, 51, 57, 65]. Lastly, as discussed in Section 2.2, given a sender-anonymous channel, timing analysis of messages with designated recipients can be mitigated using existing techniques [53, 57].

It is trivial for an active adversary that controls the platform to deny service to arbitrary users by not delivering messages. In future work, it may be valuable to provide a mechanism for honest users to provably expose such misbehavior, but in this work we leave platform-mounted denial-of-service (DoS) attacks out of scope. On the other hand, we do want to protect against user-mounted DoS attacks, in which a malicious user can interact with an honest platform to deny service to other users, as in the griefing attack.

Overview. We will now provide an overview of Orca's design by stepping through a series of strawman constructions.

Sender-specific one-time use access tokens. Instead of having all senders authenticate by reusing the same shared access token, the recipient can deal unique access tokens to each sender. Reusing a sender-specific token allows linking by the platform, so these tokens will necessarily be one-time use only. We outline a version of this approach that is taken by the Pond messaging system [46,47].

On registration, recipients register a key k to a pseudorandom function F, e.g. HMAC, with the platform. Recipients distribute one-time use tokens of the form $(x, y = F(k,x))$ for random values x to senders. The platform verifies these tokens using k and the recipient can identify senders since they know to whom they dealt (x,y). A sender's tokens are refreshed in the normal exchange of messages. Now a recipient can block by reporting the unused tokens of a sender to the platform; the platform tracks these tokens along with previously spent tokens for a recipient in a strikelist and rejects incoming messages that authenticate with struck tokens. The platform's strikelist grows unbounded as more messages are sent, but this cost can be managed by scheduled key rotations.

This blocklisting approach improves significantly over sealed sender as it effectively removes the griefing attack vector, however it does not address the concerns around leakage during initialization: the recipient still initially distributes the access tokens over non-sender-anonymous channels to senders, revealing to the platform a small set of possible senders for future messages. A different approach is needed to provide stronger sender anonymity with non-interactive initialization.

Group signatures. A promising starting point for sender-anonymous blocklisting with non-interactive initialization is *group signatures*, a well-studied cryptographic primitive [7,9,12,18,24]. Group signature schemes allow users to sign messages anonymously on behalf of a group whose membership is controlled by a *group manager*. Signatures appear anonymous to everyone except to a special *opening authority* who has the ability to deanonymize the signer and revoke their signing ability.

Our next strawman solution has the platform maintain a separate group signature scheme for each registered user, where the user is the opening authority and the platform is the group manager. A sender registers with the platform under the desired recipient's group signature scheme. The sender sends their message along with a signature on the message under the recipient's group to the platform. The platform then verifies the anonymized signature. For blocklisting, we use a group signature scheme that supports *verifier-local revocation* [15]. This means that the recipient can revoke senders by communicating *only* with the platform (i.e., verifier).

This strawman provides effective sender attribution and blocklisting. It also allows senders to acquire group signature credentials without previous interaction with the recipient. However, messages to a recipient can be attributed by the platform to the set of users that registered under the recipient's group signature scheme, so we do not achieve our stronger anonymity goal. Furthermore, existing group signatures that meet our requirements use expensive bilinear pairing operations, adding on to the efficiency concerns of managing a separate scheme for each registered user.

We resolve these issues by proposing a new type of group signature that introduces two novel features. The first is support for *multiple opening authorities*. This will dispense with the per-recipient group signature schemes and the need to register separately for each recipient that you wish to send to. The second feature is *keyed-verification*, in which we observe that the platform is also the only verifier. Removing public verifiability improves efficiency of client-side operations.

This new group signature, presented in Section 4, makes up the core of Orca. However even with our optimizations, e.g., keyed-verification, the group signature approach incurs significant computational cost, in particular for the platform, owing to the use of verifier-local revocation: verifying a signature incurs work linear in the size of the recipient's blocklist.

Hybrid: Group signature with one-time tokens. This leads us to our final construction which combines the use of group signatures for non-interactive initialization with one-time use tokens for efficient authentication of subsequent messages. Here, the group signature is used to allow the sender to acquire its first batch of tokens from the platform. The main contribution of this approach is a new protocol for allowing the platform to dispense tokens on behalf of the recipient. This is challenging because the platform should not be able to link newly minted tokens to a sender, but it must provide a way for the recipient to learn to whom new tokens were dealt (for future sender attribution). We construct this protocol by adapting techniques from blinded issuance of anonymous credentials [22]. After this (relatively) expensive initialization procedure, users exchange new tokens in the normal flow of conversation and the system enjoys all the efficiency benefits of the token-based protocol. We describe Orca's one-time token extension in Section 5.

4 Orca's Group Signature

Our main construction is based on a novel group signature scheme. In this section, we will introduce our new group signature abstraction, describe how to use it to construct an outsourced blocklisting protocol, and lastly provide an instantiation of such a group signature,

4.1 Group Signature Syntax and Security

Group signatures [24] allow users to sign messages anonymously on behalf of a group. The basic setting is as follows. The membership of a group is coordinated by a *group manager*, with whom users register with in order to join the group. Additionally, anonymous group signatures can be opened (traced) to identify the signing user in the group by a designated *opening authority*.

We make use of three extensions to the basic group signature setting.

(1) *Verifier local revocation*: A group signature supporting revocation allows the opening authority to additionally revoke the signing ability of group members. *Verifier local* revocation means that to revoke a member, the opening authority need only communicate a revocation message to verifying parties (as opposed to both verifying parties and group members); revocation does not affect the way group members sign messages.

(2) *Multiple opening authorities*: An opening authority is created through registration with the group manager. Group members sign messages designated to one of many opening authorities, and only the opening authority that a signature is designated to is able to open the signature to the signer's identity. Revocation is handled separately per opening authority, meaning a group member may be able to sign messages designated for some opening authorities, but be revoked from signing messages to others.

(3) *Keyed verification*: Verification of group signatures can only be completed by a secret key owned by the group manager and shared to verifying parties. This is particularly useful in cases where the group manager is the only party verifying signatures and allows for more efficient schemes than those that achieve public verifiability.

Verifier local revocation has been previously studied [15], but the other two extensions are novel to the best of our knowledge. The model and following security definitions for our new setting are derived from [9, 15].

Syntax. A multi-opener, keyed-verification group signature scheme GS is run between three types of participating parties: (1) users U that join the group and sign messages, (2) opening authorities OA that can trace signatures to signers, and (3) a group manager GM to coordinate registration and perform verification. It consists of the following algorithms:

- $pp \leftarrow\$ \mathsf{GS.Setup}(\lambda)$: The setup algorithm defines the public parameters pp. We will assume pp is available to all algorithms, and all parties have assurance it was created correctly.
- $(gmpk, gmsk) \leftarrow\$ \mathsf{GS.Kg}^{pp}_{\mathrm{GM}}()$: The key generation algorithm is run by the group manager to generate a public key $gmpk$ and secret key $gmsk$.
- $\mathsf{GS.JoinU}^{pp}_{\mathrm{U}} \leftrightarrow \mathsf{GS.IssueU}^{pp}_{\mathrm{GM}}$: Group registration is an interactive protocol implemented by GS.JoinU and GS.IssueU run between a user and the group manager, respectively. If execution is successful, the user will receive a public, secret key pair (upk, usk) and the group manager will receive upk, else both parties receive $\perp$. If the protocol accepts, the group manager will store upk in a global registration table and reject duplicate upk registrations.
- $\mathsf{GS.JoinOA}^{pp}_{\mathrm{OA}} \leftrightarrow \mathsf{GS.IssueOA}^{pp}_{\mathrm{GM}}$: Opening authority registration is an interactive protocol run between a prospective opening authority and the group manager. If execution is successful, the opening authority will receive a public, secret key pair $(oapk, oask)$ and the group manager will receive and store $oapk$ in the registration table, else both parties receive $\perp$.
- $\sigma \leftarrow\$ \mathsf{GS.Sign}^{pp}_{\mathrm{U}}(usk, gmpk, oapk, m)$: The signing algorithm is run by a group member to produce a group signature σ on a message m designated for opening authority $oapk$.
- $upk \leftarrow \mathsf{GS.Open}^{pp}_{\mathrm{OA}}(oask, m, \sigma)$: The opening algorithm is run by an opening authority to learn the identity of the signing user upk, and returns $\perp$ upon failure.
- $\tau_R \leftarrow\$ \mathsf{GS.Revoke}^{pp}_{\mathrm{OA}}(oask, upk)$: The revocation algorithm is run by an opening authority to create a revocation token τ_R for a user upk. The opening authority sends the revocation token to the group manager who includes it in a revocation list RL used for verification.
- $b \leftarrow \mathsf{GS.Ver}^{pp}_{\mathrm{GM}}(gmsk, oapk, RL, m, \sigma)$: The verification algorithm is run by the group manager to determine if an input signature σ and m are valid under a designated opening authority $oapk$ and revocation list RL.

As mentioned, we assume some global registration table that contains all user public keys upk and opening authority public keys $oapk$ that succeed registration. In practice, such a table might be implemented with a public key infrastructure (PKI) supporting key transparency audits [54] allowing it be hosted by the untrusted platform. Additionally, for simplicity, we may drop the executing party from the subscript and the public parameters from the superscript if their use is clear from context.

Correctness and security notions. We extend the standard notions of correctness and security from [9, 15]. Here, we describe correctness and then the three security properties: anonymity, traceability, and non-frameability. The properties are formalized via security games involving an adversary, but we defer the formal definitions to Appendix D.

The *correctness* property concerns signatures generated by honest group members. An honestly generated signature should pass verification under all honestly generated revocation lists that do not include a revocation token for the signing user created by the designated opening authority. An honestly generated signature should also be opened to the correct signing user by the designated opening authority.

The *anonymity* property (see Figure 11 for full security game) captures that an adversary without access to the designated opening authority's key should not be able to determine the signer of a signature among unrevoked group members. The adversary has the power of an actively malicious group manager and may adaptively compromise group members and opening authorities. More specifically, we target *CCA-selfless-anonymity* [12] meaning signatures are not anonymous to the signer (selfless) and the adversary has access to an opening oracle throughout the security game (CCA). We consider rogue key attacks, allowing the adversary to create public keys for corrupted parties, but require the adversary to prove knowledge of secret keys. We model this, for simplicity, by asking the adversary to produce the secret key for generated public keys following the knowledge of secret key model of [11], which can be instantiated with extractible proofs of knowledge. We also provide an extension of our anonymity game to capture anonymity of revocation tokens (in addition to signatures) that is, to our knowledge, the first definitional attempt at doing so.

Traceability (Figure 12)nsures that every signature that passes verification can be opened by the designated opening authority to a registered user. Traceability necessarily considers an adversary that does not control the group manager since it is trivial for the group manager to craft signatures for unregistered public keys. However, traceability is accompanied by *non-frameability* (Figure 13) which ensures that it is not possible to forge a signature that opens to an honest user; non-frameability considers a stronger adversary that controls the group manager as in anonymity.

Bilinear pairing groups. Our construction will make use of bilinear pairing groups for which we will use the following notation. (1) Groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are cyclic groups of prime order p. (2) Group element g_1 is a generator of $\mathbb{G}_1$, g_2 is a generator of $\mathbb{G}_2$. (3) Pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a computable map with the following properties: *Bilinearity*: $\forall\ u \in \mathbb{G}_1, v \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$, and *Non-degeneracy*: $e(g_1, g_2) \neq 1$. We assume an efficient setup algorithm that on input security parameter λ, generates a bilinear group, $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathcal{G}(\lambda)$, where $|p| = \lambda$.

4.2 Outsourced Blocklisting from Group Signatures

Given a keyed-verification, multi-opener group signature with verifier-local revocation, we build our core protocol, detailed in Figure 3. The platform plays the role of the group manager. Users register with the platform as both a user of the group and as an opening authority, receiving keys $(usk_i, oask_i)$. For user i to send a message to user j, assume for now that user i has user j's public keys $(upk_j, oapk_j)$. We will describe how user i obtains these keys shortly.

User i signs their message with usk_i under the group signature scheme designating $oapk_j$ as the opening authority. The platform verifies the anonymous group signature against user j's revocation list, and if it verifies, delivers the message and signature to user j, who can then identify the sender, upk_i, by opening the signature. Users can blocklist a sender upk_i to the platform by generating a revocation token under their opening authority key $oask_j$ and sending it to the platform. Anonymity of the group signature and revocation tokens ensure that the platform does not learn sender identity information from messages or from the blocklist; and traceability and non-frameability ensure recipients will be able to properly attribute received messages to a sender.

To achieve our stronger sender anonymity goal, user i must be able to read the public key information of user j needed to start a conversation *without* revealing their own identity to the platform. Since public key information is not sensitive, the platform can provide unrestricted access to PKI lookups that do not require user authentication. Note that the platform can observe the *number* of lookups to a recipient's public key, but learns no information on which users are making those lookups. We discuss how the platform can restrict access to resources and maintain anonymity in Section 8.

4.3 Construction of Group Signature

Our group signature follows closely the "certified signature" recipe that many group signatures take [36]. In this recipe, the group manager registers users by certifying their public key $Y = g^y$; the user's group key is made up of their secret identity key y along with the group manager's certificate t. To sign a message under the group, the user encrypts their public key to the opening authority creating an *identity ciphertext* where Z is the opening authority's encryption key.

$$ct_{id} \leftarrow (g_1^{\alpha_{ct}}, YZ^{\alpha_{ct}}) \qquad \alpha_{ct} \leftarrow\!\!\$\ \mathbb{Z}_p$$

They then prove in zero knowledge that they have a certificate from the group manager on the same public key that is enclosed in the ciphertext *and* that they know the secret key associated to it. The signature is verified by verifying the zero knowledge proof and can be opened by the opening authority simply by decrypting the identity ciphertext.

This recipe naturally extends to support a scheme with multiple opening authorities. The identity ciphertext is encrypted using the public key of the designated opening authority.

Supporting verifier-local revocation. An opening authority registers with two keys: (1) an encryption key $(z, Z = g_1^z)$, and (2) a revocation key $(w, W = g_1^w)$, where $oapk = (W, Z)$. We have described how a user with identity key $(y, Y = g_1^y)$ encrypts their public key Y to the opening authority. To revoke a user's signing ability, the opening authority constructs a user-specific *revocation token* as the Diffie-Hellman value between the user's public key and their own revocation key, $\tau_R = Y^w$. Intuitively, these revocation tokens are anonymous since a Diffie-Hellman value looks random to a verifier that

Protocol 1: Orca Outsourced Blocklisting Protocol

Setup:

(1) Public parameters for the group signature scheme are generated, $pp \leftarrow\$ \mathsf{GS.Setup}(\lambda)$.

(2) The platform initializes its state as the group manager of the group signature scheme.

  (a) $(gmpk, gmsk) \leftarrow\$ \mathsf{GS.Kg}^{pp}_{\mathrm{GM}}()$

  (b) $T_U \leftarrow [\cdot]$: Table tracking user public keys.

  (c) $T_R \leftarrow [\cdot]$: Table tracking user revocation tokens.

Registration:

(1) User registers with platform to acquire group signature signing key with which to send messages, $\mathsf{GS.JoinU}^{pp}_{\mathrm{U}} \leftrightarrow \mathsf{GS.IssueU}^{pp}_{\mathrm{GM}}$. User stores usk and platform stores upk.

(2) User registers as opening authority and generates keys with which to receive messages, $\mathsf{GS.JoinOA}^{pp}_{\mathrm{OA}} \leftrightarrow \mathsf{GS.IssueOA}^{pp}_{\mathrm{GM}}$. User stores $oask$ and platform stores $oapk$.

(3) Platform stores public keys in $T_U[upk] \leftarrow oapk$.

(4) Platform initializes empty revocation token list for user, $T_R[oapk] \leftarrow [\cdot]$.

Sending a message:

(1) [Optional] Sender anonymously requests recipient public key ($oapk$) and/or rate-limited pre-keys from platform (described in Section 8).

(2) Sender signs message specifying the recipient as the opening authority (with recipient's $oapk$), $\sigma \leftarrow\$ \mathsf{GS.Sign}^{pp}_{\mathrm{U}}(usk, gmpk, oapk, m)$. Sender sends message, signature, and recipient to platform, $(m, \sigma, oapk)$.

(3) Platform checks validity of signature against recipient's revocation list, $b \leftarrow \mathsf{GS.Ver}^{pp}_{\mathrm{GM}}(gmsk, oapk, T_R[oapk], m, \sigma)$. If $b = 1$, then platform delivers (m, σ) to recipient.

Blocklisting a user:

(1) Recipient generates and sends anonymous revocation token to platform,

  (a) $upk \leftarrow \mathsf{GS.Open}^{pp}_{\mathrm{OA}}(oask, m, \sigma)$

  (b) $\tau_R \leftarrow\$ \mathsf{GS.Revoke}^{pp}_{\mathrm{OA}}(oask, upk)$

(2) Platform adds revocation token to recipient's blocklist, $T_R[oapk] \leftarrow T_R[oapk] \cup \{\tau_R\}$.

(3) [Optional] Recipient stores identities of blocklisted senders and/or reports sender identity to platform (described in Section 8).

Figure 3: Core protocol based on group signature.

does not know the secret keys y or w.

To allow a verifier in possession of a user's revocation token to identify signatures from a user, we need something more. In addition to the identity ciphertext, the user also constructs a *revocation ciphertext* enclosing their revocation token, $\tau_R = W^y$. This "ciphertext" is constructed to be undecryptable, but includes a backdoor for testing whether a plaintext pt is enclosed (following the approach of Boneh and Shacham [15]).

$$ct_R \leftarrow (M_1^{\alpha_T}, \tau_R N_1^{\alpha_T}) \qquad \alpha_T \leftarrow\$ \mathbb{Z}_p \qquad M_1, N_1 \leftarrow\$ \mathbb{G}_1$$

The backdoor of ct_R consists of the isomorphic $\mathbb{G}_2$ elements M_2, N_2. The verifier can check whether $\hat{\tau}_R$ is enclosed in ct_R via the following test using the pairing function e:

$$e(T_2/\hat{\tau}_R, M_2) \stackrel{?}{=} e(T_1, N_2) \qquad (T_1, T_2) \leftarrow ct_R$$

The verifier performs this test for each revocation token in an opening authority's revocation list and outputs 1 if no revocation token matches and the signature's proof verifies. The signature's proof now additionally proves the well-formedness of ct_R with respect to user public key Y.

Improving efficiency with keyed-verification. A central part of the group signature is that the user must prove they have a certificate on their public key from the group manager. Creating this proof, even for certificate signatures designed for this purpose [12, 20], is relatively expensive, with known constructions requiring multiple pairings to be evaluated. In our setting, the platform plays the role of both the group manager and the sole verifier; all messages pass through the platform. This setting allows us to bring in techniques from keyed-verification anonymous credentials [22]. Specifically, during user registration, instead of receiving a signature from the group manager, users receive a MAC t on their public key from an algebraic MAC scheme; our construction uses $\mathsf{MAC}_{\mathsf{GGM}}$ from [22, 31]. Proving knowledge of a valid MAC is more efficient and, in particular, does not require pairing evaluations. The resulting proof can only be verified using the secret MAC key (held by the group manager), hence our introduction of the keyed-verification setting for group signatures (i.e., "group MACs"). This optimization limits the use of pairings in our group signature only to the revocation token tests made by the group manager during verification.

Summary. In total, our group signature is composed of three components, (1) the identity ciphertext ct_{id} enclosing the signer's public key to the opening authority, (2) the revocation ciphertext ct_R enclosing the revocation token, and (3) a zero knowledge proof π that (1) and (2) were constructed properly with knowledge of a key pair (y, Y) and a MAC t on Y. The full details of the construction are given in Figure 8 in Appendix C, and we prove security of our scheme with respect to our formal definitions of anonymity, traceability, and non-frameability in Appendix D.

As stated, every time a user sends a message, they create a group signature and the platform verifies the group signature. Even with our optimizations, this involves the platform running a verification algorithm that is linear in the size of the recipient's revocation list. We improve in the next section, extending Orca with one-time use sender tokens to make the need for a group signature a rare event.

5 Extending Orca with One-time Use Tokens

In this section, we describe how to reduce Orca's reliance on its core group signature protocol. Instead of creating and verifying a group signature for every message sent, the group signature will only be used periodically to mint new batches of one-time use sender tokens from the platform. Messages can be sent, with very little cost, by including a valid token for a recipient. Furthermore, once communication with a recipient has been established, a recipient can replenish a sender's tokens directly in a return message, avoiding the need to mint more token batches from the platform. The protocol is detailed in Figure 4.

Blinded MACs as one-time use tokens. We want that a sender can anonymously mint a batch of tokens for a recipient from the platform. The platform should not be able to link the tokens (when they are spent) to the time of minting. To realize this, we again turn to algebraic MACs used by keyed-verification anonymous credentials [22]; we use $\mathsf{MAC}_{\mathsf{GGM}}$. Each user generates a MAC secret key $sk \leftarrow (x_0, x_1) \in \mathbb{Z}_p^2$ and sends it to the platform. A valid MAC on input $\nu \in \mathbb{Z}_p$ is of the form,

$$t \leftarrow (u_0, u_1 = u_0^{x_0 + x_1 \nu}) \qquad u_0 \leftarrow\!\!\$\, \mathbb{G}_1\,.$$

To blindly evaluate a MAC on input ν, a user generates a random ElGamal key pair $(\gamma, D = g_1^\gamma)$ and encrypts g_1^ν to D,

$$ct = (ct_1 = g_1^r, ct_2 = g_1^\nu D^r) \qquad r \leftarrow\!\!\$\, \mathbb{Z}_p\,.$$

The user blinds a batch of inputs $[\nu]_i$ in this manner, creates a group signature σ over $[ct]_i$ designating the recipient as the opening authority, and then sends $(\sigma, [ct]_i, D)$ to the platform. The platform verifies the group signature under the recipient's revocation list, and if verification succeeds, proceeds with the blind evaluation using the recipient's MAC secret key. By the homomorphic properties of ElGamal, the platform can maul ct to form ct' as an encryption of a valid MAC on ν without ever learning anything about ν,

$$ct' = (ct_1^{x_1 \cdot b} g_1^{r'}, ct_2^{x_1 \cdot b} u_0^{x_0} D^{r'}) \quad u_0 \leftarrow g_1^b \qquad b, r' \leftarrow\!\!\$\, \mathbb{Z}_p\,.$$

The full details of the blind MAC evaluation is given in Figure 9 of Appendix C. The user decrypts ct' to learn u_1 and stores token $\tau \leftarrow (\nu, t = (u_0, u_1))$ as the input, tag pair.

To send a message, the user sends the message to the platform along with an unused token τ for the recipient. The platform checks that the token $(\nu, t) \leftarrow \tau$ is unused, i.e., ν is not in the strikelist of used tokens for a recipient, and that the token is valid, i.e., the MAC t is valid for ν under the recipient's MAC key. If those checks pass, the platform delivers the message along with the token τ to the recipient and adds ν to the recipient's strikelist.

However, the recipient has no way identifying the sender from the token τ. The generation of τ was (necessarily) blinded to prevent linking by the platform, but that also prevents linking by the recipient.

Allowing a recipient to link tokens to senders. Senders must communicate to the recipient the unblinded inputs ν for which they are minting tokens. They do this by additionally encrypting the input ν to the recipient under the recipient's public key Z,

$$\hat{ct} = (\hat{ct}_1 = g_1^{\hat{r}}, \hat{ct}_2 = g_1^\nu Z^{\hat{r}}) \qquad \hat{r} \leftarrow\!\!\$\, \mathbb{Z}_p\,,$$

and proving in zero knowledge that the input ν enclosed in the blinded ciphertext ct is the same as that enclosed in the ciphertext $\hat{ct}$ to the recipient(details highlighted in Figure 9). The sender signs the batch of recipient ciphertexts $[\hat{ct}]_i$ under the group signature with the recipient as the designated opening authority. As before, if the signature σ verifies under the recipient's revocation list, the platform proceeds with blind evaluation, *but also* sends $(\sigma, [\hat{ct}]_i)$ to the recipient.

The recipient opens σ to the sender's identity upk, then decrypts and stores the token identifiers $[g_1^\nu]_i$. Later when a recipient receives a message and token $(\nu, t) \leftarrow \tau$ from the platform, they can link the token to the sender by looking up g_1^ν. To block a sender, the recipient generates and sends the revocation token for the sender's upk to the platform so the sender cannot mint new tokens, as well as sends the sender's remaining unused tokens $[g_1^\nu]_i$ to add to the strikelist.

Replenishing tokens directly from the recipient. The motivation for one-time use tokens was to avoid the cost of the more expensive group signature for every message. However, in some sense, the gain from not running the group signature for every message is offset by the upfront cost of generating a proof to mint each token. While there are optimizations that can be made when batching proofs in this manner [40], this is still an unsatisfying result.

The real efficiency gain from one-time use tokens is when senders can replenish their tokens directly from the recipient, without going through the blind minting process with the platform. Once two users have established sender-anonymous communication, they can use their own secret MAC keys to generate and exchange tokens directly at very little cost.

Summary. In this protocol, the core group signature is used only to initiate conversations and mint the first batch of tokens. Once conversation has been established, messages can be exchanged and tokens can be replenished at almost no cost, beyond storage. With regards to storage, users must maintain lists of unused tokens in order to send messages and identify senders of received messages. The platform also needs to maintain an ever-growing strikelist for each user; in practice, users will need to periodically rotate their keys to refresh the platform strikelist, but can ensure that they have distributed tokens for the new key prior to doing so.

Using tokens does leak some information about user communication patterns in a nuanced way. An example might be that if senders need to often mint tokens from the platform for a particular user, the platform can infer that user is not active in responding and replenishing sender tokens.

A second nuance is that in both our scheme and the token strawman [46, 47] presented in Section 3, the message ciphertext of a sender is not bound to the token. The platform can forward the sender's token to the recipient, but swap out the ciphertext, so the recipient will incorrectly attribute it to the sender. In Section 6, we discuss why the impact of such an attack is not large if the underlying E2EE protocol provides message authentication. Nevertheless, we provide a proposal for modifying our token showing protocol to bind the sender's message ciphertext using a BLS signature [14] in Appendix E.

Despite these nuances, we feel Orca with one-time use tokens represents an attractive design choice.

Protocol 2: Orca with One-time Use Tokens

Setup:

(1) Public parameters for the group signature scheme, algebraic MAC scheme, and public key encryption scheme are generated, $pp \xleftarrow{\$} \mathsf{GS.Setup}(\lambda)$, $pp_{\mathsf{M}} \xleftarrow{\$} \mathsf{MAC.Setup}(\lambda)$, $pp_{\mathsf{PKE}} \xleftarrow{\$} \mathsf{PKE.Setup}(\lambda)$.

(2) The platform initializes its state as the group manager of the group signature scheme.

- (a) $(gmpk, gmsk) \xleftarrow{\$} \mathsf{GS.Kg}^{pp}_{\mathrm{GM}}()$
- (b) $T_U \leftarrow [\cdot]$: Table storing user public keys.
- (c) $T_R \leftarrow [\cdot]$: Table storing user revocation tokens.
- (d) $T_k \leftarrow [\cdot]$: Table storing user token MAC key and encryption key.
- (e) $T_\tau \leftarrow [\cdot]$: Table storing strikelist of previously-used tokens for user.

Registration:

(1) User generates keys for protocol and initializes recipient state:

- (a) User registers with platform to acquire group signature signing key with which to send messages, $\mathsf{GS.JoinU}^{pp}_{\mathrm{U}} \leftrightarrow \mathsf{GS.IssueU}^{pp}_{\mathrm{GM}}$. User stores usk and platform stores upk.
- (b) User registers as opening authority and generates keys with which to block senders, $\mathsf{GS.JoinOA}^{pp}_{\mathrm{OA}} \leftrightarrow \mathsf{GS.IssueOA}^{pp}_{\mathrm{GM}}$.
- (c) User generates algebraic MAC key used for creating sender tokens, $(tsk, tpk) \xleftarrow{\$} \mathsf{MAC.Kg}^{pp_{\mathsf{M}}}()$, and sends both tsk and tpk to platform.
- (d) User generates keys for public key encryption scheme, $(ek, dk) \xleftarrow{\$} \mathsf{PKE.Kg}()$, stores dk and sends ek to platform.
- (e) User initializes two tables, T_x and T_x^{-1}, to identify (and blocklist) senders and their associated sender tokens.

(2) Platform stores keys and initializes table entries for user:
$T_U[upk] \leftarrow (oapk)$; $T_k[oapk] \leftarrow (tsk, tpk, ek)$
$T_R[oapk] \leftarrow [\cdot]$; $T_\tau[oapk] \leftarrow [\cdot]$

Sending a message:

(1) Sender selects unused sender token for recipient and sends message, token, and recipient, $(m, \tau, oapk)$, to platform.

(2) Platform checks if token $(x, t) \leftarrow \tau$ is valid under recipient's MAC key $(tsk, tpk, ek) \leftarrow T_k[oapk]$ and if token was not already used (i.e., is not on strikelist).
$b_1 \leftarrow \mathsf{MAC.Ver}^{pp_{\mathsf{M}}}(tsk, x, t)$
$b_2 \leftarrow (x \notin T_\tau[oapk])$
If $b_1 = 0$ or $b_2 = 0$, platform aborts.

(3) Platform adds token to strikelist, $T_\tau[oapk] \leftarrow T_\tau[oapk] \cup \{x\}$.

(4) Platform forwards message and token value, (m, x), to recipient.

(5) Recipient removes token from list of valid tokens for sender,
$T_x[T_x^{-1}[x]] \leftarrow T_x[T_x^{-1}[x]] \setminus \{x\}$; $T_x^{-1}[x] \leftarrow \perp$.

Acquiring sender tokens (from platform):

(1) [Optional] Sender anonymously requests public key information, $(oapk, tpk, ek)$, for desired recipient from platform.

(2) Sender authenticates to platform as a non-blocklisted sender for the recipient using a group signature.

- (a) Sender signs set of recipient ciphertexts $[\hat{ct}]_i$ (constructed in (3)) with recipient as opening authority, and sends $(\sigma, oapk)$ to platform, $\sigma \xleftarrow{\$} \mathsf{GS.Sign}^{pp}_{\mathrm{U}}(usk, gmpk, oapk, [\hat{ct}]_i)$.
- (b) Platform checks validity of signature against recipient's revocation list, $b \leftarrow \mathsf{GS.Ver}^{pp}_{\mathrm{GM}}(gmsk, oapk, T_R[oapk], [\hat{ct}]_i, \sigma)$. If $b = 0$, then platform aborts.

(3) Sender engages in token generation protocol with platform.

- (a) Sender samples m inputs, $[x]_i^m \xleftarrow{\$} \mathsf{MAC.In}(\lambda)^m$.
- (b) Sender encrypts inputs to recipient, $\hat{ct}_i \xleftarrow{\$} \mathsf{PKE.Enc}(ek, x_i)$.
- (c) Sender and platform engage in MAC blind evaluation for each token, $\mathsf{MAC.BlindInp}^{pp_{\mathsf{M}}}(tpk, x_i) \leftrightarrow \mathsf{MAC.BlindEv}^{pp_{\mathsf{M}}}(tsk)$, for recipient keys $(tsk, tpk, ek) \leftarrow T_k[oapk]$. Sender also sends proof that the input used in the MAC protocol is properly well-encrypted in the ciphertext to the recipient:
$$\pi_i \xleftarrow{\$} \mathsf{NiZK}\{x_i : \mathsf{MAC.BlindInp}^{pp_{\mathsf{M}}}(tpk, x_i) \wedge ct_i = \mathsf{PKE.Enc}^{pp_{\mathsf{PKE}}}(ek, x_i)\}$$
If π_i does not verify, platform aborts the blind MAC protocol.
- (d) If blind MAC protocol succeeds, sender receives MAC t_i as output and stores token, $\tau_i \leftarrow (x_i, t_i)$.

(4) Platform sends $(\sigma, [\hat{ct}]_i^m)$ to recipient.

(5) Recipient stores tokens to later identify sender.

- (a) Recipient traces sender, $upk \leftarrow \mathsf{GS.Open}^{pp}_{\mathrm{OA}}(oask, [\hat{ct}]_i, \sigma)$.
- (b) Recipient decrypts token ciphertexts and stores tokens.
$x_i \leftarrow \mathsf{PKE.Dec}^{pp_{\mathsf{PKE}}}(dk, \hat{ct}_i)$
$T_x[upk] \leftarrow T_x[upk] \cup [x_1, \ldots, x_m]$; $T_x^{-1}[x_i] \leftarrow upk$

Acquiring sender tokens (from recipient):

(1) Recipient samples m inputs, $(x_1, \ldots, x_m) \xleftarrow{\$} \mathsf{MAC.In}(\lambda)^m$, and MACs them, $t_i \leftarrow \mathsf{MAC.Ev}^{pp_{\mathsf{M}}}(tsk, x_i)$.

(2) Recipient sends tokens $\tau_i \leftarrow (x_i, t_i)$ to sender associated with upk out-of-band or via secure channel.

(3) Recipient stores tokens to later identify sender.
$T_x[upk] \leftarrow T_x[upk] \cup \{x_1, \ldots, x_m\}$; $T_x^{-1}[x_i] \leftarrow upk$

Blocklisting a user:

(1) Recipient looks up sender identity associated with token, $upk \leftarrow T_x^{-1}[x]$, and generates revocation token, $\tau_R \xleftarrow{\$} \mathsf{GS.Revoke}^{pp}_{\mathrm{OA}}(oask, upk)$. Recipient sends revocation token along with list of remaining sender tokens for sender to platform, $(x_1, \ldots, x_m) \leftarrow T_x[upk]$.

(2) Platform updates blocklist state by adding revocation token to blocklist and remaining tokens to strikelist.
$T_R[oapk] \leftarrow T_R[oapk] \cup \{\tau_R\}$
$T_\tau[oapk] \leftarrow T_\tau[oapk] \cup \{x_1, \ldots, x_m\}$

Figure 4: Hybrid protocol based on group signature and tokens.

6 Composition with an E2EE Protocol

The main security properties of an E2EE messaging protocol are *message confidentiality* and *message authentication*. Modern forms of message confidentiality include forward secrecy and post-compromise security which ensure that, even in the event of key compromise, previous message content and future message content (after recovery) are not leaked, respectively [25]. Message authentication ensures that messages accepted by the recipient were those encrypted by the sender. A third property, *repudiability*, requires that the authentication mechanism cannot help non-conversation participants verify message authorship, even if secrets from a conversation participant are leaked [16]. For our setting, we will also require the E2EE messaging protocol to be *sender-anonymous*, meaning ciphertexts do not leak any information about the sender, which can be achieved using encapsulation as in sealed sender.

Orca composes with an E2EE messaging protocol to further provide anonymous, outsourced blocklisting (see Section 3). Public keys for Orca may be distributed using the same mechanism used to distribute public keys for the E2EE messaging protocol. Similar to E2EE messaging, to prevent ghost key attacks by a malicious PKI, in which a user's key is replaced by one owned by the adversary, users are expected to perform manual verification of key fingerprints out-of-band or perform periodic auditing of the PKI [54]. Without this assurance, ghost key attacks against Orca result in a break in anonymity, as the adversary can open group signatures using the ghost key. Of course, using Orca does not increase the damage of such attacks: such an adversary can read encrypted messages and break anonymity by subverting the E2EE.

In basic Orca (Figure 3), E2EE ciphertexts are sent along with a group signature over the ciphertext, and when extended with one-time sender tokens (Figure 4), E2EE ciphertexts are sent along with a token produced from a token minting protocol authenticated with a group signature. The composition preserves the message confidentiality and authentication properties of the underlying E2EE protocol: Orca composes generically with the E2EE ciphertexts and does not make further use of the message plaintext. However, Orca necessarily weakens sender-anonymity and repudiability to support blocklisting by a third-party (the platform).

With regards to anonymity, a necessary leakage of the outsourced blocklisting setting is that a ciphertext leaks (to the platform) whether or not the sender is present on the designated recipient's blocklist. Basic Orca meets this minimum leakage, following directly from the anonymity and revocation anonymity security properties of the group signature. Orca extended with one-time tokens leaks more: platform-assisted token minting leaks how many tokens for a recipient are minted, and blocking reveals how many valid tokens remain for the blocked sender. In addition to the anonymity properties of the group signature, achieving only this level of leakage relies on (1) randomly chosen MAC inputs, (2) security of blind MAC evaluation, (3) confidentiality of the recipient ElGamal ciphertexts, and (4) zero knowledge of the well-formedness proof. We believe it is unlikely the additional leakage of token counts leads to damaging inference attacks, especially considering token counts are further obscured by tokens replenished directly by the recipient.

The minimum weakening to repudiability for the outsourced blocklisting setting is that the platform can at most verify authorship to *some* registered member of the platform, even with compromised secrets. However, our group signature construction does not meet this weakened notion; the platform and the recipient can together provide proof of authorship of a message for a sender. Future work may adapt techniques from deniable signatures (c.f., [63]) to recover repudiability.

Lastly, outsourced blocklisting requires sender attribution: messages delivered to recipients can be correctly attributed to a sender. Basic Orca achieves sender attribution following directly from the traceability security property of the group signature. The extension with one-time tokens achieves sender attribution additionally relying on the soundness of the well-formedness proof of recipient token-tracing ciphertexts.

We also note an optional non-frameability property: a malicious platform should not be able to frame a user as being a sender for a ciphertext they did not create. We do not see this property as security-critical for outsourced blocklisting. A break in non-frameability allows a platform to deliver ciphertexts that are misattributed, however, due to the message authentication property of the underlying E2EE protocol, these ciphertexts will not be accepted by the recipient. The recipient may choose to block the misattributed sender, mistakenly thinking they are spamming malformed ciphertexts. We view this as a special (slightly more damaging) case of a platform-mounted DoS attack, which is not a goal of Orca to defend against. Nevertheless, basic Orca does prevent this attack due to the non-frameability security property of the group signature. Orca with one-time tokens can be extended with token-binding (see Appendix E) to achieve non-frameability relying on the soundness of the blind MAC evaluation proof and the unforgeability of the token-binding signature.

Formal analyses. As mentioned, we provide in Appendix D formal definitions and security analyses for our group signature, the core underlying component of Orca. These analyses do not cover the one-time token extension, nor the security of the composition informally discussed above. Developing formal models suitable for analysis of these higher level primitives remains an open problem. Our initial attempts suggest that this will be challenging, as it seems to require extending existing (already complex) confidentiality and authenticity models for messaging (e.g., [5,10,25,41,58]) to model sender anonymity, token distribution, blocklist maintenance, etc. An ideal functionality based approach may provide an alternative tack, though any resulting functionality will also be complex (possibly as complicated as our protocols).

7 Implementation and Evaluation

This section aims to evaluate the feasibility of deploying Orca at scale. Specifically, we answer the following questions:

- *Client costs*: What are the processing and storage costs that Orca incurs on user clients?
- *Platform costs*: What are the processing and storage costs incurred on the platform? What throughput (user activity) can be reasonably supported given these costs?
- *Bandwidth costs*: How large are Orca protocol messages? What additional networking costs does Orca introduce?

To answer these questions, we provide a prototype library in Rust of our group signature and token-based scheme. Our implementation is over the BLS12-381 pairing-friendly elliptic curve and uses the `zexe/algebra` Rust pairing library [17]. We instantiate the proofs of knowledge using standard Sigma protocols of discrete logarithm relations [18] made non-interactive using the Fiat-Shamir transform [33]. Our security proofs (see Appendix D) rely on a simulation-extractability property of the zero knowledge proofs which has been shown to hold in the algebraic group model [34] for the knowledge of discrete logarithm relation [3,35]; we believe these techniques can be readily extended to the discrete logarithm relations used in this work. Our implementation consists of less than 1400 lines of code and is available open source [2].

The experiments, including the microbenchmarks given in Figure 5, were performed using a `c5.12xlarge` Amazon EC2 virtual machine with 24 cores and 96 GB of memory running Ubuntu Server 20.04 LTS as the platform and desktop client (single-core) and on a Google Pixel device running Android 9 as the mobile client. The platform is implemented using an in-memory Redis database for storing revocation blocklists and token strikelists.

When evaluating Orca, recall that users can replenish their token supply directly from the recipient provided there is back and forth communication. Thus, we make the distinction between "initialization costs" of minting an initial token batch from the platform and the "steady-state costs" that occur when tokens are replenished directly from the communicating partner. We expect the majority of user communication to be in steady-state where costs are low.

Client costs. Clients must store, for each of their communicating partners, two lists of unused tokens, one for sending messages and one for identifying received messages. These tokens are not large (240B) and the lists can remain small as they can be replenished on next communication. Say a user has 200 communication partners and stores 20 tokens per list. This setup would incur $\sim$ 1MB for the client.

The bulk of the processing costs incurred by Orca are concentrated at initialization when a client mints an initial batch of tokens to start a conversation. On a mobile client, minting an initial batch of tokens takes $\sim$ 150 ms for the group signature and an additional $\sim$ 100 ms for each token in the batch (see Figure 5). This means it takes around 1 second for a sender to mint 10 tokens. While these costs are significant, we stress that a user only needs to mint enough tokens to initiate a conversation and await a response. If a response from a recipient is delayed, more tokens can be minted as needed. Once a conversation with back-and-forth communication is established, the amortized steady-state cost of sending a message is in creating a new token to replenish the recipient, which is done at very little cost ($\sim$ 10 ms) — approximately the same as sealed sender.

Platform costs. The platform stores per-recipient revocation blocklists and token strikelists. The revocation lists are on the order of 100B / revoked user; e.g., a recipient that has blocked 100 users would require a revocation list of size 10KB to be stored. We do not anticipate revocation lists to grow too large, since the platform has other mechanisms to ban users globally (see Section 8). In any case, a platform can impose limits on the size of revocation lists if necessary.

The per-recipient strikelists would grow in size with every message a user sends (32B / spent token). One can use Bloom filters or other data structures to compress the size of the strikelist as well as enforce periodic key rotations to reset its size. If each user sends $\sim$ 100 messages per day and token keys are rotated every two weeks, the platform can store a strikelist of $\sim$ 5KB per user with a false positive rate of 10^{-6}. Note the false positive rate can be traded off with storage size; messages that get rejected due to false positives will result in an error returned to the anonymous sender, who may resend with a different token.

The processing costs of the platform are similarly dominated by the token mint requests for initializing conversation as opposed to send requests during steady-state conversation. A request to mint a batch of 10 tokens given a recipient blocklist size of 100 takes $\sim$ 200 ms to complete whereas a send request is just a simple algebraic MAC verification and strikelist lookup taking < 1 ms (see Figure 5).

Figure 6 demonstrates these workloads are easily parallelizable to achieve high levels of throughput. In this experiment, we run the platform with one million users, each with a blocklist of size 100 and a strikelist of size 1400 (100 messages/-day/two weeks), and measure the rate at which the platform can process requests for different levels of hardware parallelism. We do not implement the Bloom filter optimization, so the Redis database stores $\sim$ 50KB per user (50GB total), which can still easily fit in memory. The computationally expensive mint requests parallelize with essentially no loss, reaching a rate of 80 requests (for 10 token batches) per second on 24 cores. The inexpensive send requests also parallelize but top out at around 30000 requests per second on 12 cores, which is bottlenecked by the operation throughput of a single Redis database and can be unblocked via a different database setup if needed (e.g. through sharding). The achieved bottlenecked throughput already demonstrates feasibility.

[2] https://github.com/nirvantyagi/orca

Operation		Platform	User (Desktop client)		User (Mobile client)	
			Sender	Recipient	Sender	Recipient
Sealed sender		–	0.50 (0.02)	0.50 (0.02)	6.6 (0.2)	6.6 (0.2)
Orca	mint tokens with group signature	11.2 (0.2)	10.8 (0.1)	9.7 (0.2)	131.7 (0.8)	117 (2)
	+ cost per token minted	7.60 (0.09)	8.50 (0.08)	0.30 (0.01)	105.2 (0.9)	3.3 (0.1)
	+ cost per blocked user	1.70 (0.04)	–	–	–	–
	send message with token*	0.30 (0.01)	0.80 (0.02)	–	10.0 (0.2)	–

*Steady-state cost of sending a message with a token that includes cost of replenishing one token

Figure 5: Processing time (ms) microbenchmarks of user and platform operations for Orca compared to sealed sender. Mean time is given with standard deviations shown in parentheses. Dashes indicate an operation that has negligible cost (e.g., a table lookup).

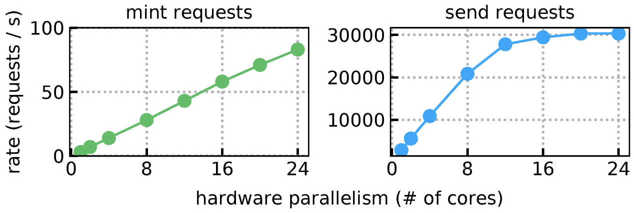

Figure 6: Platform request throughput for different levels of hardware parallelism over a one million user deployment with blocklists of size 100 and strikelists of size 1400. Each mint request corresponds to a request to mint a batch of 10 tokens.

Bandwidth costs. Minting a token requires sending the group signature (1.6KB) and exchanging proofs for each token to be minted (0.7KB / token). These costs extend to the recipient who receives the signature and also a ciphertext for each token minted (0.2KB / token). Apart from these initialization costs, the steady-state bandwidth costs of sending a message, once again, compare quite favorably with sealed sender. In the steady state, the amortized bandwidth overhead of sending a message would be two tokens (240B / token) — the token being spent *and* the token being created to replenish the recipient. Thus we can achieve amortized per-message overheads of only 30B compared to sealed sender (450B / message).

8 Further Extensions

Backwards unlinkability for revocation tokens. A drawback of verifier-local revocation is that whenever a new revocation token is provided, the platform can replay the history of messages to link which ones were sent by the newly blocked sender. To prevent such leakage one can take the approach of [55] to rotate revocation keys in set epochs. Naively, this requires recipients to resupply their entire list of revocation tokens; future work may try to incorporate techniques from updatable encryption [13] to provide more efficient epoch transitions.

Credential expiry and global banning. Per-recipient blocklists are not a substitute for platform-wide banning of abusive users. The platform must maintain some mechanism for banning accounts in the case of identified user abuse, e.g., through user reports [30, 38, 63] or account compromise. This can be done by enforcing periodic credential expiration, by for example, rotating the platform's group manager key. Users must retrieve a new MAC on their public key, at which point, the platform can choose to deny their request.

Sybil resistance and account recovery. Outsourced blocklisting works by blocking a public key, not an identity. If malicious users are able to easily send messages under new public keys, either by registering with many accounts or continually rotating an account key after they are blocklisted, then our blocklisting protocol will be of little use. Signal ties accounts to phone numbers to mitigate the ability to easily register new accounts. On the other hand, rotating an account key is a legitimate operation that may need to be taken after account compromise or device loss. Blocking accounts with suspicious key rotation behavior or rate-limiting account recovery are possible mitigations.

Rate-limited resources. In Signal, in addition to needing the recipient's long-lived identity public key, senders also need to pull a one-time use recipient "pre-key" which is used in the initial key agreement protocol to provide forward secrecy properties. Recipients store some number of pre-keys with the platform and replenish them as needed. If a recipient's pre-keys run out, then conversations are initiated without the pre-key leading to weaker forward secrecy. To prevent malicious users from exhausting a recipient's pre-key supply, these resources can be protected while preserving anonymous authentication using anonymous rate-limiting techniques [19].

9 Related Work

Anonymous credentials. Anonymous credentials [20] allow a user to present a cryptographic token proving some specific statement about their identity (e.g., their authorization to send messages to a particular recipient), without revealing anything else about their identity. A problem with anonymous credentials in our setting is that they are — by design — not attributable. While the server processing messages can verify the sender is authorized, the recipient cannot identify the sender. This means there is no way for the server to block the sender in the future, even if some revocation mechanism for the credentials did exist.

A notable design contrast to general-purpose anonymous credential schemes is Privacy Pass [28], which offers single use credentials that encode only one bit — "I am authorized."

Privacy Pass mints tokens using a verifiable oblivious pseudorandom function [42, 43], which is more efficient than our approach of blind MACs [22], but does not provide the algebraic structure needed to prove relations on the input. We need this property to encrypt the input to the recipient to allow linking of tokens. Blind MACs have been previously suggested for use as one-time tokens [50] and have also been recently proposed as part of Signal's new proposal for private group messaging [23].

Anonymous blacklisting. Anonymous blacklisting [39, 59, 60] systems cover a variety of cryptographic techniques. In general, these systems allow a user to authenticate anonymously to third parties in such a way that the third party can block them from subsequent authentications if they misbehave. In some systems, this blocking ability takes the form of an additional trusted third party that can de-anonymize users much like a group signature. In others, every time a user authenticates they provide a fresh anonymous cryptographic token derived from their identity and a proof that the current blacklist contains no tokens generated by their own keys. Such systems are cryptographically expensive, requiring work linear in the blacklist (for the sender). Moreover, much of the overhead across both settings comes from providing anonymity from the third party. Our setting differs in that the sender need not be anonymous (and in fact, should be identifiable) to the party *adding* to the blacklist (i.e., the recipient), but only be anonymous to the party *filtering* on the blacklist (i.e., the platform).

Abuse reporting in E2EE messaging. A complementary line of work [30, 32, 38, 63] considers reporting abusive content sent over an encrypted channel. These systems allow the recipient to verifiably reveal the content of a message to the platform to enable content moderation. They allow attribution of message content to a sender for a known sender identity. They do not allow the attribution of a malformed message with unknown sender as in the griefing attack we describe.

Metadata-private messaging. A number of messaging systems have been proposed that provide strong metadata-privacy even against strong network adversaries [4, 6, 26, 27, 45, 48, 51, 57, 62, 64, 65]. These systems incur significant costs on their users, e.g. to send and receive messages at frequent intervals. These costs may dwarf the costs of the types of abuse that Orca aims to prevent. Despite this, a subclass of these systems that could still make use of Orca for blocklisting are based on anonymous broadcasting [26, 44, 51, 57, 65]. Anonymous broadcasts can be converted to a sender-anonymous messaging service by having a messaging service collect, filter, and deliver the broadcast messages with designated recipients.

10 Conclusion

This paper explores the tensions between abuse mitigation and sender-anonymity in E2EE messaging. We highlighted several issues with Signal's sealed sender feature, including weak anonymity set guarantees and vulnerability to griefing attacks.

Our solution, Orca, allows recipients to register privacy-preserving blocklists with the platform. Without learning the sender's identity, the platform can check that the sender is not on the blocklist and that the recipient will be able to verify their identity. We introduced a new type of group signature tailored to Orca's needs and propose a hybrid scheme that uses tokens to amortize the bandwidth and computational costs of group signatures.

Acknowledgments

The authors would like to thank Trevor Perrin for discussions about the griefing attack and the proposed mitigations. Nirvan Tyagi was supported by a Facebook Graduate Research Fellowship. Julia Len was supported by the NSF Graduate Research Fellowship under Grant No. NSF DGE-1650441. This work was supported in part by NSF grant 2120651.

References

[1] Battery Historian. https://github.com/google/battery-historian, 2017.

[2] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle diffie-hellman assumptions and an analysis of DHIES. In *CT-RSA*, 2001.

[3] Michel Abdalla, Fabrice Benhamouda, and Philip MacKenzie. Security of the J-PAKE password-authenticated key exchange protocol. In *IEEE S&P*, 2015.

[4] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. Mcmix: Anonymous messaging via secure multiparty computation. In *USENIX Security*, 2017.

[5] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the signal protocol. In *EUROCRYPT*, 2019.

[6] Sebastian Angel and Srinath T. V. Setty. Unobservable communication over fully untrusted infrastructure. In *OSDI*, 2016.

[7] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT*, 2003.

[8] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT*, 2006.

[9] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA*, 2005.

[10] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In *CRYPTO*, 2017.

[11] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *PKC*, 2003.

[12] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, 2004.

[13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In *CRYPTO*, 2013.

[14] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, 2001.

[15] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In *CCS*, 2004.

[16] Nikita Borisov, Ian Goldberg, and Eric A. Brewer. Off-the-record communication, or, why not to use PGP. In *WPES*, 2004.

[17] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *IEEE S&P*, 2020.

[18] Jan Camenisch. *Group signature schemes and payment systems based on the discrete logarithm problem*. PhD thesis, ETH Zurich, Zürich, Switzerland, 1998.

[19] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *CCS*, 2006.

[20] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, 2004.

[21] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *CRYPTO*, 2006.

[22] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. In *CCS*, 2014.

[23] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. In *CCS*, 2020.

[24] David Chaum and Eugène van Heyst. Group signatures. In *EUROCRYPT*, 1991.

[25] Katriel Cohn-Gordon, Cas J. F. Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *IEEE EuroS&P*, 2017.

[26] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *IEEE S&P*, 2015.

[27] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *CCS*, 2010.

[28] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018.

[29] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security*, 2004.

[30] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In *CRYPTO*, 2018.

[31] Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. Message authentication, revisited. In *EUROCRYPT*, 2012.

[32] Facebook. Messenger Secret Conversations technical whitepaper. https://fbnewsroomus.files.wordpress.com/2016/07/messenger-secret-conversations-technical-whitepaper.pdf, 2017.

[33] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[34] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *CRYPTO*, 2018.

[35] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed elgamal encryption in the algebraic group model. In *EUROCRYPT*, 2020.

[36] Jens Groth. Fully anonymous group signatures without random oracles. In *ASIACRYPT*, 2007.

[37] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In *CRYPTO*, 2017.

[38] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In *CRYPTO*, 2017.

[39] Ryan Henry and Ian Goldberg. Formalizing anonymous blacklisting systems. In *IEEE S&P*, 2011.

[40] Ryan Henry and Ian Goldberg. Batch proofs of partial knowledge. In *ACNS*, 2013.

[41] Joseph Jaeger and Igors Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. In *CRYPTO*, 2018.

[42] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *ASIACRYPT*, 2014.

[43] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In *EuroS&P*, 2016.

[44] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. Atom: Horizontally scaling strong anonymity. In *SOSP*, 2017.

[45] Albert Kwon, David Lu, and Srinivas Devadas. XRD: scalable messaging system with cryptographic privacy. In *NSDI*, 2020.

[46] Adam Langley. Pond. https://github.com/agl/pond, 2016.

[47] Adam Langley and Trevor Perrin. Replacing group signatures with HMAC in Pond. https://moderncrypto.org/mail-archive/messaging/2014/000409.html, 2016.

[48] David Lazar, Yossi Gilad, and Nickolai Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In *OSDI*, 2018.

[49] Benoît Libert, Thomas Peters, and Moti Yung. Short group signatures via structure-preserving signatures: Standard model security from simple assumptions. In *CRYPTO*, 2015.

[50] Isis Agora Lovecruft and Henry de Valence. HYPHAE: Social secret sharing. https://patternsinthevoid.net/hyphae/hyphae.pdf, 2017.

[51] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew K. Miller. Honeybadgermpc and asynchromix: Practical asynchronous MPC and its application to anonymous communication. In *CCS*, 2019.

[52] Joshua Lund. Technology preview: Sealed sender for Signal. `https://signal.org/blog/sealed-sender/`, 2017.

[53] Ian Martiny, Gabriel Kaptchuk, Adam Aviv, Dan Roche, and Eric Wustrow. Improving Signal's sealed sender. In *NDSS*, 2021.

[54] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: bringing key transparency to end users. In *USENIX Security Symposium*, 2015.

[55] Toru Nakanishi and Nobuo Funabiki. Verifier-local revocation group signature schemes with backward unlinkability from bilinear maps. In *ASIACRYPT*, 2005.

[56] Trevor Perrin and Moxie Marlinspike. The double ratchet algorithm. `https://signal.org/docs/specifications/doubleratchet/`, 2016.

[57] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The loopix anonymity system. In *USENIX Security*, 2017.

[58] Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In *CRYPTO*, 2018.

[59] Patrick P. Tsang, Man Ho Au, Apu Kapadia, and Sean W. Smith. Blacklistable anonymous credentials: blocking misbehaving users without ttps. In *CCS*, 2007.

[60] Patrick P. Tsang, Apu Kapadia, Cory Cornelius, and Sean W. Smith. Nymble: Blocking misbehaving users in anonymizing networks. *IEEE Trans. Dependable Sec. Comput.*, 2011.

[61] Yiannis Tsiounis and Moti Yung. On the security of elgamal based encryption. In *PKC*, 1998.

[62] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *SOSP*, 2017.

[63] Nirvan Tyagi, Paul Grubbs, Julia Len, Ian Miers, and Thomas Ristenpart. Asymmetric message franking: Content moderation for metadata-private end-to-end encryption. In *CRYPTO*, 2019.

[64] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *SOSP*, 2015.

[65] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *OSDI*, 2012.

A Griefing Attack on Sealed Sender

We identify and implement a griefing attack against Signal's sealed sender protocol. An attacker in possession of a recipient's access key can spam the recipient with untraceable messages, causing the recipient's system to suffer battery drain and to consume bandwidth.

Attack vector. The attack takes advantage of the fact that the platform cannot check for malformed sealed messages. Our proof-of-concept attack simply modifies the Signal client to modify the triple (pk_e, ct_{id}, ct_{ss}) by replacing pk_e with a new, random value pk_f. To the platform this is indistinguishable from a legitimate sealed sender message, but the recipient's decryption will fail when trying to decrypt ct_{id} and cannot recover any information about the sender. Our modification causes the recipient's decryption to fail early. While technically one could force the recipient to perform more cryptographic steps, this would have small impact on the efficacy of the attack.

This approach required changing only two lines of code in the Signal Desktop client. We also wrote a small script to automate sending messages via the client.

Attack efficacy. We performed some measurements to assess whether the griefing attack can be used, particularly, to drain a target's battery. In our experiments, we used as attacker our modified Signal Desktop application on a MacBook Pro 2017 machine running macOS Mojave using a 2.5 GHz Intel Core i7. We used as a stand-in for victim recipient an unmodified Signal Android application (version 4.54.3) on a Google Pixel phone running Android version 9. We used the Android Battery Historian tool [1] to inspect the effect of our attack on battery drainage. It reports the battery level rounded to the nearest percent.

In our experiments we only interacted with the Signal platform and with researcher devices. We purposefully experimented only with very low volume attacks in order to ensure we did not burden the Signal platform, and confirmed ahead of time with members of the Signal team that our experiments would not be problematic. In summary, the platform and its users were not negatively affected by our experiments.

We measured the rate of change in battery level per hour when sending one malformed sealed message every 1, 2, 5, or 10 seconds. As a baseline comparison, we also measured the rate of battery drainage when no messages were sent. Each of the four sending rates were measured over a period of 2 hours, while the baseline was measured over a period of 11 hours; the phone discharges slowly at rest so an extended measurement period was needed for the baseline. Before each experiment, the recipient phone was rebooted and charged to full capacity. During each experiment, the phone used its mobile data for network connectivity and was otherwise idle.

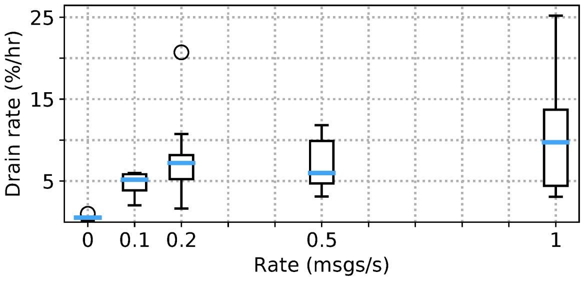

Figure 7: Battery drain rate of griefing attack for various rates of sending, $x \in \{0, 0.1, 0.2, 0.5, 1\}$ / second. The box plot shows the variability of drain rates over trials, with the range, quartiles and median denoted by the whiskers, box, and line, respectively (outliers marked separately).

In the baseline case, where the phone received no malicious messages, the battery level dropped by only 0.45 levels per hour (dropping the battery by only 7% in 11 hours). In comparison, the drop rates were 4.11, 5.37, 5.84, and 6.88 levels per hour when sending a message once every 10s, 5s, 2s, and 1s. Thus even the slowest attack rate speeds up battery drain by 9x; for one message a second it is 15x. We show a boxplot of these measurements in Figure 7.

The attack also consumes recipient bandwidth (which could be costly if they pay for data service per byte): at one message per second, the Signal Android application received 1.13 MB/hour, while as a baseline it receives 0.94 KB/hour.

A real attacker can of course trivially increase attack volume up to any general rate limiting enforced by the platform. While it is not public if Signal rate limits clients (and we did not want to stress test it), we believe even modest increases to the volume will allow draining batteries quickly. While battery drain rates will vary significantly based on target handset and other factors, we believe our proof-of-concept evidences sufficient impact on a victim to be a concern.

Mitigation options for victims. The receiver's Signal client gives no obvious visual indication that messages are being received and filtered. To learn of message filtering, a user would have to inspect the client's debug logs, making the attack essentially invisible for the majority of users. Even if detected, there are no particularly good ways to prevent the griefing attack.

The victim can rotate their access key ak and attempt to redistribute a new ak' to their communicating partners. If the attacker is not able to get access to the new access key, the attack will be stopped by the platform and no messages will reach the victim's client. But since the attack leaves no information about which of the victim's communicating partners is responsible, the victim can only make a guess as to whom they should block.

This issue might lead people to only add a few, highly trusted contacts. But this degrades anonymity significantly, since as discussed in Section 2.2, the platform knows that a sealed sender must be one of the recipient's contacts.

B Proof Preliminaries

Notation. We use $x \leftarrow y$ and $x \leftarrow \mathsf{Eval}()$ to denote assigning the value of y and the evaluation of Eval to variable x. If Eval uses random coins, we instead denote $x \xleftarrow{\$} \mathsf{Eval}$. For finite set Y, we denote $x \xleftarrow{\$} Y$ as sampling a random value from the set. We denote a dictionary D initialized as $[\cdot]$ to store key-value pairs (k,v). Adding or updating a value v for key k is denoted as $D[k] \leftarrow v$. A table T is a special use of a dictionary in which values are added in sequence with incrementing keys. We denote appending a value v to a table with $T \hookleftarrow v$. We will allow for membership queries on dictionaries of the form $k \in D$, $v \in D$, and $(k,v) \in D$, also allowing for wildcard queries of the form $(k,*) \in D$.

To model interactive protocols between two parties, we define an algorithm for each party that takes an incoming message and a current state, and returns an outgoing message, an updated state, and a decision in $\{\mathtt{accept}, \mathtt{reject}, \mathtt{cont}\}$. If the decision is $\mathtt{accept}$, the output of the protocol for the party will be stored in the state.

We also define notation for prime-order cyclic groups. We assume an efficient setup algorithm that on input security parameter λ, generates a group description, $(p, \mathbb{G}, g) \leftarrow \mathcal{G}(\lambda)$, where for group order p, $|p| = \lambda$, and g is a canonical generator. The group operation is denoted by multiplication.

Bilinear pairing groups. We follow the notation of [14]. (1) Groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are cyclic groups of prime order p. (2) Group element g_1 is a generator of $\mathbb{G}_1$, g_2 is a generator of $\mathbb{G}_2$. (3) Pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a computable map with the following properties: *Bilinearity*: $\forall\ u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u,v)^{ab}$, and *Non-degeneracy*: $e(g_1, g_2) \neq 1$. We assume an efficient setup algorithm that on input security parameter λ, generates a bilinear group, $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathcal{BG}(\lambda)$, where $|p| = \lambda$.

B.1 Computational Assumptions

Discrete log assumption. The discrete log assumption is defined by the security game $\mathrm{G}^{\mathrm{dl}}_{\mathbb{G},p,\mathcal{A}}(\lambda)$ in which an adversary is tasked with finding the discrete log of a random group element. The advantage of an adversary is defined as $\mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G},p,\mathcal{A}}(\lambda) = \Pr[\mathrm{G}^{\mathrm{dl}}_{\mathbb{G},p,\mathcal{A}}(\lambda) = 1]$. We will make use of the discrete log assumption in $\mathbb{G}_1$ of the bilinear pairing groups, which is one of the assumptions made by external Diffie-Hellman (XDH).

Decisional Diffie-Hellman assumption. The decisional Diffie-Hellman (DDH) assumption is defined by the security game $\mathrm{G}^{\text{ddh-}b}_{\mathbb{G},p,\mathcal{A}}(\lambda)$ in which an adversary is tasked with distinguishing between a triple of random group elements and a random Diffie-Hellman triple. The advantage of an adversary is defined as $\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},p,\mathcal{A}}(\lambda) = \left|\Pr[\mathrm{G}^{\text{ddh-1}}_{\mathbb{G},p,\mathcal{A}}(\lambda) = 1] - \mathrm{G}^{\text{ddh-0}}_{\mathbb{G},p,\mathcal{A}}(\lambda) = 1]\right|$. We will make use of the DDH assumption in $\mathbb{G}_1$ of the bilinear pairing groups, which is one of the assumptions made by external Diffie-Hellman (XDH).

Decision linear assumption. The decision linear (DLIN) assumption is defined by the security game $\mathrm{G}^{\text{dlin-}b}_{\mathbb{G},p,\mathcal{A}}(\lambda)$ in which an adversary is tasked with distinguishing between a set of three random group elements along with those same three values taken to different random exponents and a set where the last group element is not taken to a random exponent but the sum of the previous two exponents. The decision linear assumption is considered to hold even in groups where DDH is easy and thus is thought to hold in pairing groups even when the associated group elements in the paired group are revealed. In our version of the game, we explicitly return the group elements in $\mathbb{G}_2$ since we will need to make use of them in our reductions. If we used a pairing type with an efficiently computable isomorphism then we wouldn't need to this change. This variant is sometimes referred to as the external decision linear assumption (XDLIN) [49]. The advantage of an adversary is defined as $\mathbf{Adv}^{\mathrm{dlin}}_{\mathbb{G},p,\mathcal{A}}(\lambda) = \left|\Pr[\mathrm{G}^{\text{dlin-1}}_{\mathbb{G}_1,\mathbb{G}_2,p,\mathcal{A}}(\lambda) = 1] - \mathrm{G}^{\text{dlin-0}}_{\mathbb{G}_1,\mathbb{G}_2,p,\mathcal{A}}(\lambda) = 1]\right|$.

```
Game $\mathrm{G}^{\mathrm{dl}}_{\mathbb{G},p,\mathcal{A}}(\lambda)$
$g \xleftarrow{\$} \mathbb{G}$
$x \leftarrow \mathbb{Z}_p$
$x' \xleftarrow{\$} \mathcal{A}(g^x, g)$
Return $x == x'$
```

```
Game $\mathrm{G}^{\text{ddh-}b}_{\mathbb{G},p,\mathcal{A}}(\lambda)$
$g \xleftarrow{\$} \mathbb{G}$
$(\alpha, \beta, \gamma) \xleftarrow{\$} \mathbb{Z}_p$
$C_0 \leftarrow g^{\gamma}$ ; $C_1 \leftarrow g^{\alpha\beta}$
$b' \xleftarrow{\$} \mathcal{A}(g^{\alpha}, g^{\beta}, C_b, g)$
Return $b'$
```

```
Game $\mathrm{G}^{\text{dlin-}b}_{\mathbb{G}_1,\mathbb{G}_2,p,\mathcal{A}}(\lambda)$
$g_1 \xleftarrow{\$} \mathbb{G}_1$ ; $g_2 \xleftarrow{\$} \mathbb{G}_2$
$(\alpha, \beta, \gamma) \xleftarrow{\$} \mathbb{Z}_p$
$(m, n, l) \xleftarrow{\$} \mathbb{Z}_p$
$m_1 \leftarrow g_1^m$ ; $m_2 \leftarrow g_2^m$
$n_1 \leftarrow g_1^n$ ; $n_2 \leftarrow g_2^n$
$l_1 \leftarrow g_1^l$ ; $l_2 \leftarrow g_2^l$
$C_0 \leftarrow l_1^{\gamma}$ ; $C_1 \leftarrow l_1^{\alpha+\beta}$
$b' \xleftarrow{\$} \mathcal{A}(m_1, n_1, l_1, m_1^{\alpha}, n_1^{\beta}, C_b, m_2, n_2, l_2)$
Return $b'$
```

B.2 Public-key Encryption

Syntax and correctness. A public key encryption scheme PKE is a tuple of algorithms ($\mathsf{PKE.Setup}$, $\mathsf{PKE.Kg}$,$\mathsf{PKE.Enc}$, $\mathsf{PKE.Dec}$). The setup algorithm produces the public parameters for the scheme, $pp \leftarrow\$\, \mathsf{PKE.Setup}(\lambda)$. The key generation algorithm outputs a public encryption key and a secret decryption key, $(ek, dk) \leftarrow\$\, \mathsf{PKE.Kg}^{pp}()$. The encryption algorithm produces a ciphertext on an input message, $ct \leftarrow\$\, \mathsf{PKE.Enc}^{pp}(ek, m)$, and the decryption algorithm decrypts the ciphertext to retrieve the enclosed message, $m \leftarrow\$\, \mathsf{PKE.Dec}^{pp}(dk, ct)$. Correctness dictates that $\mathsf{PKE.Dec}(dk, \mathsf{PKE.Enc}(ek, m)) = m$ for all valid key pairs (ek, dk) and messages m in the message space.

Indistinguishability under chosen-plaintext attacks (IND-CPA). Indistinguishability under chosen-plaintext attacks (IND-CPA) for a public key encryption scheme PKE is defined by the security game $\mathrm{G}^{\text{indcpa-}b}_{\mathsf{PKE},\mathcal{A}}(\lambda)$ in which an adversary is tasked with distinguishing the decryption of a challenge ciphertext to one of two distinct self-chosen plaintexts. The advantage of an adversary is defined as $\mathbf{Adv}^{\text{indcpa}}_{\mathsf{PKE},\mathcal{A}}(\lambda) = \left|\Pr[\mathrm{G}^{\text{indcpa-1}}_{\mathsf{PKE},\mathcal{A}}(\lambda) = 1] - \mathrm{G}^{\text{indcpa-0}}_{\mathsf{PKE},\mathcal{A}}(\lambda) = 1]\right|$.

ElGamal construction. We provide pseudocode for the ElGamal public key encryption scheme ElG below. It is IND-CPA-secure under the DDH assumption [61].

```
Game $\mathrm{G}^{\text{indcpa-}b}_{\mathsf{PKE},\mathcal{A}}(\lambda)$
$pp \leftarrow\$\, \mathsf{PKE.Setup}(\lambda)$
$(pk, sk) \leftarrow\$\, \mathsf{PKE.Kg}^{pp}()$
$b' \leftarrow\$\, \mathcal{A}^{\textsc{Enc}}(pk)$
Return $b'$

$\textsc{Enc}(m_0, m_1)$
Return $\mathsf{PKE.Enc}(m_b)$
```

```
$\mathsf{ElG.Setup}(\lambda)$
$(p, g, \mathbb{G}) \leftarrow\$\, \mathcal{G}(\lambda)$
$pp \leftarrow (p, g, \mathbb{G})$
Return $pp$

$\mathsf{ElG.Kg}^{pp}()$
$x \leftarrow\$\, \mathbb{Z}_p$
Return $(g^x, x)$

$\mathsf{ElG.Enc}^{pp}(ek, m)$
$r \leftarrow\$\, \mathbb{Z}_p$
$ct \leftarrow (g^r, mek^r)$
Return $ct$

$\mathsf{ElG.Dec}^{pp}(dk, ct)$
$(ct_1, ct_2) \leftarrow ct$
Return $ct_2/ct_1^{dk}$
```

B.3 Non-interactive Zero Knowledge Proofs and Signatures of Knowledge

We define a non-interactive proof system NiZK over an efficiently computable relation $\mathcal{R}$ defined over pairs (x, w) where x is called the *statement* and w is called the *witness*. Let $\mathcal{L}$ be the language consisting of statements in $\mathcal{R}$.

A non-interactive proof system NiZK is made up of the following algorithms. The setup algorithm produces the public parameters for execution, $pp \leftarrow\$\, \mathsf{NiZK.Setup}(\lambda)$. The proving algorithm takes a witness and statement and produces a proof, $\pi \leftarrow\$\, \mathsf{NiZK.Prove}^{pp}(w, x)$. The verification algorithm verifies a proof for a statement, $b \leftarrow \mathsf{NiZK.Ver}^{pp}(x, \pi)$. We further extend the notion of a non-interactive proof system to a signature of knowledge proof system SoK by modifying the proving and verification algorithms to support binding a message [21, 37]. A signature of knowledge is similar to a digital signature in that a message can only be validly signed with respect to a statement by a party with knowledge of a witness. The signing algorithm and signature verification algorithm additionally take a message m as input, $\mathsf{NiZK.Prove}^{pp}(w, x, m)$ and $\mathsf{NiZK.Ver}^{pp}(x, m, \pi)$.

The below definitions will apply to both a non-interactive proof system and to a signature of knowledge proof system. Extensions to the non-interactive proof system definitions introduced for signatures of knowledge are highlighted.

Completeness. A proof system is *complete* if given a true statement, a prover with a witness can convince the verifier. We will make use of a proof system with perfect completeness. A proof system has *perfect completeness* if for all $(x, w) \in \mathcal{R}$ and all m in the message space,

$$\Pr[\mathsf{NiZK/SoK.Ver}(x, m, \mathsf{NiZK/SoK.Prove}(w, x, m)) = 1] = 1 \;.$$

Knowledge soundness. A proof system is computationally *knowledge sound* if whenever a prover is able to produce a valid proof, it is possible to extract a valid witness from the prover's internal transcript. The prover's internal transcript, denoted by τ, contains the description of the prover algorithm and input along with any random choices made. Knowledge soundness is defined by the security game $\mathrm{G}^{\text{sound}}_{\mathsf{NiZK},\mathcal{A},\mathcal{X}}(\lambda)$ in which an adversary is tasked with finding a verifying statement and proof for which the extractor does not extract a valid witness. The advantage of an adversary is defined as $\mathbf{Adv}^{\text{sound}}_{\mathsf{NiZK},\mathcal{A},\mathcal{X}}(\lambda) = \Pr[\mathrm{G}^{\text{sound}}_{\mathsf{NiZK},\mathcal{A},\mathcal{X}}(\lambda) = 1]$.

Simulation extractability. Simulating a proof for a false statement might jeopardize the soundness of the proof system. It may be possible for an adversary to modify the proof into another proof for a false instance. This scenario is common in security proofs of cryptographic schemes, in which case it is desireable to have some sort of non-malleable property that prevents this type of break in soundness even in the presence of simulated proofs.

A proof system is *simulation extractable* if even after seeing many simulated proofs, whenever a prover produces a new proof, it is possible to extract a valid witness from their internal transcript. Simulation extractability is defined by the security game

$\mathrm{G}^{\text{simext}}_{\mathsf{NiZK},\mathcal{A},\mathcal{X},\mathcal{S}}(\lambda)$ in which an adversary is given access to a simulation oracle and tasked with finding a verifying statement and proof for which the extractor does not extract a valid witness. The advantage of an adversary is defined as $\mathbf{Adv}^{\text{simext}}_{\mathsf{NiZK},\mathcal{A},\mathcal{X},\mathcal{S}}(\lambda) = \Pr[\mathrm{G}^{\text{simext}}_{\mathsf{NiZK},\mathcal{A},\mathcal{X},\mathcal{S}}(\lambda) = 1]$.

Observe that simulation extractability implies knowledge soundness, since the games are identical if the simulation extractability adversary does not use its simulation oracle.

Zero knowledge. A proof system is computationally *zero-knowledge* if a proof does not leak any information besides the truth of a statement. Zero knowledge is defined by the security game $\mathrm{G}^{\text{zk-}b}_{\mathsf{NiZK},\mathcal{A},\mathcal{S}}(\lambda)$ in which an adversary is tasked with distinguishing between proofs generated from a valid witness and simulated proofs generated without a witness. The advantage of an adversary is defined as $\mathbf{Adv}^{\text{zk}}_{\mathsf{NiZK},\mathcal{A},\mathcal{S}}(\lambda) = \left|\Pr[\mathrm{G}^{\text{zk-1}}_{\mathsf{NiZK},\mathcal{A},\mathcal{S}}(\lambda) = 1] - \mathrm{G}^{\text{zk-0}}_{\mathsf{NiZK},\mathcal{A},\mathcal{S}}(\lambda) = 1]\right|$, with respect to simulator algorithm $\mathcal{S}$.

```
Game $\mathrm{G}^{\text{sound}}_{\mathsf{NiZK},\mathcal{A},\mathcal{X}}(\lambda)$ ; $\mathrm{G}^{\text{sound}}_{\mathsf{SoK},\mathcal{A},\mathcal{X}}(\lambda)$
$pp \leftarrow\!\!\$\ \mathsf{NiZK/SoK.Setup}(\lambda)$
$(x, \pi, m) \leftarrow\!\!\$\ \mathcal{A}(pp)$
$w \leftarrow \mathcal{X}(\tau_{\mathcal{A}})$
$b \leftarrow \mathsf{NiZK/SoK.Ver}(x, m, \pi)$
Return $(x, w) \notin \mathcal{R} \wedge b$
```

```
Game $\mathrm{G}^{\text{zk-}b}_{\mathsf{NiZK},\mathcal{A},\mathcal{S}}(\lambda)$ ; $\mathrm{G}^{\text{zk-}b}_{\mathsf{SoK},\mathcal{A},\mathcal{S}}(\lambda)$
$pp_1 \leftarrow\!\!\$\ \mathsf{NiZK/SoK.Setup}(\lambda)$
$(pp_0, \xi) \leftarrow\!\!\$\ \mathcal{S}.\mathsf{Setup}(\lambda)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Prove}}(pp_b)$
Return $b'$

$\textsc{Prove}(x, w, m)$
Require $(x, w) \in \mathcal{R}$
$\pi_1 \leftarrow\!\!\$\ \mathsf{NiZK/SoK.Prove}(x, w, m)$
$\pi_0 \leftarrow\!\!\$\ \mathcal{S}.\mathsf{Prove}(\xi, x, m)$
Return $\pi_b$
```

```
Game $\mathrm{G}^{\text{simext}}_{\mathsf{NiZK},\mathcal{A},\mathcal{X},\mathcal{S}}(\lambda)$ ; $\mathrm{G}^{\text{simext}}_{\mathsf{SoK},\mathcal{A},\mathcal{X},\mathcal{S}}(\lambda)$
$pp_1 \leftarrow\!\!\$\ \mathsf{NiZK/SoK.Setup}(\lambda)$
$(pp_0, \xi) \leftarrow\!\!\$\ \mathcal{S}.\mathsf{Setup}(\lambda)$
$(x, \pi, m) \leftarrow\!\!\$\ \mathcal{A}^{\textsc{SimProve}}(pp_b)$
$w \leftarrow \mathcal{X}(\tau_{\mathcal{A}})$
$b \leftarrow \mathsf{NiZK/SoK.Ver}(x, m, \pi)$
Return $(x, w) \notin \mathcal{R} \wedge (x, \pi, m) \notin \mathcal{Q} \wedge b$

$\textsc{SimProve}(x, m)$
$\pi \leftarrow\!\!\$\ \mathcal{S}.\mathsf{Prove}(\xi, x, m)$
$\mathcal{Q} \leftarrow\!\!\cup\ (x, \pi, m)$
Return $\pi$
```

C Full Construction Details

Here we provide the full details of our group signature and one-time use token constructions. Figure 8 gives the pseudocode for the group signature and Figure 9 gives our modified version of the blind MAC evaluation from [22] (relevant to the token protocol).

Our construction makes use of a number of proofs of knowledge of various standard discrete log relationships. Our security proofs are independent of the choice of zero knowledge proof system with which to instantiate the scheme, relying only on the simulation-extractability and zero-knowledge properties described above. In our implementation, we evaluate the classic proof system based on use of Sigma protocols, the building blocks of which are outlined by Camenisch [18]. Our proofs of knowledge are made non-interactive using the Fiat-Shamir heuristic in which the Sigma protocol commitments and proof statement are hashed to get the challenge for the Sigma protocol. The signature of knowledge algorithms are instantiated with the Fiat-Shamir heuristic by additionally passing m into the hash function along with the commitments and statement when generating a challenge. It has been shown that the simulation-extractibility property holds in the algebraic group model [34] for the knowledge of discrete logarithm relation [3, 35]. We believe the techniques used [35] can be applied to show simulation-extractability of the discrete logarithm relations used in this work.

We model interactive algorithms using the syntax from previous work [7, 9]. Each algorithm takes an incoming message and a current state, and returns an outgoing message, an updated state, and a decision in $\{\texttt{accept}, \texttt{reject}, \texttt{cont}\}$.

D Group Signature Security Proofs

Here we provide our formal definitions along with proofs of security and discussion of alternative security targets.

D.1 Correctness

Correctness is defined by the game $\mathrm{G}^{\text{corr}}_{\mathsf{GS},\mathcal{A}}$ shown in Figure 10 and explained below. We define the advantage of adversary $\mathcal{A}$ as:

$$\mathbf{Adv}^{\text{corr}}_{\mathsf{GS},\mathcal{A}}(\lambda) = \Pr\left[\mathrm{G}^{\text{corr}}_{\mathsf{GS},\mathcal{A}}(\lambda) = 1\right].$$

We say that a verifier-local revocable, keyed-verification, multi-opener group signature GS is *correct* if $\mathbf{Adv}^{\text{corr}}_{\mathsf{GS},\mathcal{A}}(\lambda) = 0$ for any adversary $\mathcal{A}$ and any $\lambda \in \mathbb{N}$. Note that the adversary is not computationally restricted.

In the correctness game, the adversary can query $\textsc{AddU}$ and $\textsc{AddOA}$ oracles to register new users and opening authorities, each running their respective join/issue interactive protocol with the group manager; the adversary is given the public and secret key of the registered party. The adversary can also query $\textsc{Revoke}$ to add user i to opening authority j's revocation list; the

```
$\Pi_{\mathsf{GS}}.\mathsf{Setup}(\lambda)$
$(p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,g_1,g_2,e) \leftarrow \mathcal{BG}(\lambda)$
$\alpha \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $h_1 \leftarrow g_1^{\alpha}$
$REG_U, REG_{OA} \leftarrow [\cdot]$
$pp \leftarrow (p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e,g_1,h_1,g_2)$
$pp_{\mathsf{M}} \leftarrow (p,\mathbb{G}_1,g_1,h_1)$
Return $pp$

$\Pi_{\mathsf{GS}}.\mathsf{JoinU}_{\mathrm{U}}^{pp}(gmpk, m_{in} : st)$
If $m_{in} == \bot$ do
    $y \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $Y \leftarrow g_1^{y}$
    $m_{out} \leftarrow Y$ ; $st \leftarrow (Y,y)$
    Return $(m_{out}, \mathtt{cont}, st)$
$(t,\pi,Y_r) \leftarrow m_{in}$
$(u_0,u_1) \leftarrow t$ ; $(y,Y) \leftarrow st$ ; $(X_1,C_{\tilde{x}_0}) \leftarrow gmpk$
$b \leftarrow \mathsf{NiZK}_{\mathcal{R}_1}.\mathsf{Ver}(\pi,(g_1,h_1,u_0,u_1,X_1,C_{\tilde{x}_0},Y,Y_r))$
$usk \leftarrow (t,y)$ ; $upk \leftarrow Y$
$st \leftarrow (upk, usk)$
If $b == 1$ then return $(\bot, \mathtt{accept}, st)$
Return $(\bot, \mathtt{reject}, st)$

$\Pi_{\mathsf{GS}}.\mathsf{IssueU}_{\mathrm{GM}}^{pp}(gmsk, m_{in} : st)$
$Y \leftarrow m_{in}$
$st \leftarrow Y$
$t \leftarrow \mathsf{MAC}_{\mathsf{GGM}}.\mathsf{GroupElemEv}^{pp_{\mathsf{M}}}(gmsk, Y)$
    $(x_0,x_1,\tilde{x}_0) \leftarrow gmsk$
    $r \leftarrow\!\!\$\ \mathbb{Z}_p$
    $u_0 \leftarrow g_1^{r}$ ; $u_1 \leftarrow (g_1^{x_0}Y^{x_1})^{r}$
    Return $(u_0,u_1)$
$Y_r \leftarrow Y^{r}$ ; $X_1 \leftarrow h_1^{x_1}$ ; $C_{\tilde{x}_0} \leftarrow g_1^{x_0}h_1^{\tilde{x}_0}$
$\pi \leftarrow\!\!\$\ \mathsf{NiZK}_{\mathcal{R}_1}.\mathsf{Prove}($
    $(x_0,x_1,\tilde{x}_0,r),(g_1,h_1,u_0,u_1,X_1,C_{\tilde{x}_0},Y,Y_r))$
$m_{out} \leftarrow (t,\pi,Y_r)$
Return $(m_{out}, \mathtt{accept}, st)$

$\Pi_{\mathsf{GS}}.\mathsf{JoinOA}_{\mathrm{OA}}^{pp}(gmpk, m_{in} : st)$
If $m_{in} == \bot$ do
    $w \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $W \leftarrow g_1^{w}$
    $z \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $Z \leftarrow g_1^{z}$
    $m_{out} \leftarrow (W,Z)$ ; $oapk \leftarrow (W,Z)$ ; $oask \leftarrow (w,z)$
    $st \leftarrow (oapk, oask)$
    Return $(m_{out}, \mathtt{cont}, st)$
Return $(\bot, \mathtt{accept}, st)$

$\Pi_{\mathsf{GS}}.\mathsf{IssueOA}_{\mathrm{GM}}^{pp}(gmsk, m_{in} : st)$
$(W,Z) \leftarrow m_{in}$
$st \leftarrow (W,Z)$
Return $(\top, \mathtt{accept}, st)$

$\Pi_{\mathsf{GS}}.\mathsf{Kg}_{\mathrm{GM}}^{pp}()$
$(gmpk, gmsk) \leftarrow\!\!\$\ \mathsf{MAC}_{\mathsf{GGM}}.\mathsf{Kg}^{pp_{\mathsf{M}}}()$
    $x_0 \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $x_1 \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $\tilde{x}_0 \leftarrow\!\!\$\ \mathbb{Z}_p$
    $X_1 \leftarrow h_1^{x_1}$ ; $C_{\tilde{x}_0} \leftarrow g_1^{x_0}h_1^{\tilde{x}_0}$
    $pk \leftarrow (X_1, C_{\tilde{x}_0})$ ; $sk \leftarrow (x_0,x_1,\tilde{x}_0)$
    Return $(pk, sk)$
Return $(gmpk, gmsk)$

$\Pi_{\mathsf{GS}}.\mathsf{Sign}_{\mathrm{U}}^{pp}(usk, gmpk, oapk, m)$
$(t,y) \leftarrow usk$ ; $(u_0,u_1) \leftarrow t$
$(X_1,C_{\tilde{x}_0}) \leftarrow gmpk$ ; $(W,Z) \leftarrow oapk$
$\alpha_{ct},\alpha_u,\alpha_y,\alpha_T,\beta \leftarrow\!\!\$\ \mathbb{Z}_p^4$
$r_m \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $r_n \leftarrow\!\!\$\ \mathbb{Z}_p$
$M_1 \leftarrow g_1^{r_m}$ ; $M_2 \leftarrow g_2^{r_m}$ ; $N_1 \leftarrow g_1^{r_n}$ ; $N_2 \leftarrow g_2^{r_n}$
$u_0' \leftarrow u_0^{\beta}$ ; $u_1' \leftarrow u_1^{\beta}$
$\tau_R \leftarrow W^{y}$
$ct_1 \leftarrow g_1^{\alpha_{ct}}$ ; $ct_2 \leftarrow g_1^{y}Z^{\alpha_{ct}}$
$T_1 \leftarrow M_1^{\alpha_T}$ ; $T_2 \leftarrow \tau_R N_1^{\alpha_T}$
$C_y \leftarrow u_0'^{y}h_1^{\alpha_y}$ ; $C_u \leftarrow u_1' g_1^{\alpha_u}$ ; $V \leftarrow g_1^{-\alpha_u}X_1^{\alpha_y}$
$\pi \leftarrow\!\!\$\ \mathsf{SoK}_{\mathcal{R}_2}.\mathsf{Prove}((y,\alpha_y,\alpha_u,\alpha_{ct},\alpha_T,r_m,r_n),$
        $(g_1,h_1,u_0',X_1,C_y,V,W,Z,ct_1,ct_2,M_1,M_2,N_1,N_2,T_1,T_2),m)$
$\sigma \leftarrow (u_0',C_y,C_u,V,ct_1,ct_2,M_1,M_2,N_1,N_2,T_1,T_2,\pi)$
Return $\sigma$

$\Pi_{\mathsf{GS}}.\mathsf{Open}_{\mathrm{OA}}^{pp}(oask, gmpk, m, \sigma)$
$(w,z) \leftarrow oask$ ; $W \leftarrow g_1^{w}$ ; $Z \leftarrow g_1^{z}$ ; $(X_1,C_{\tilde{x}_0}) \leftarrow gmpk$
$(u_0,C_y,C_u,V,ct_1,ct_2,M_1,M_2,N_1,N_2,T_1,T_2,\pi) \leftarrow \sigma$
Require $\mathsf{SoK}_{\mathcal{R}_2}.\mathsf{Ver}($
    $(g_1,h_1,u_0,X_1,C_y,V,W,Z,ct_1,ct_2,M_1,M_2,N_1,N_2,T_1,T_2),m,\pi)$
$upk \leftarrow ct_2/ct_1^{z}$
Return $upk$

$\Pi_{\mathsf{GS}}.\mathsf{Revoke}_{\mathrm{OA}}^{pp}(oask, upk)$
$(w,z) \leftarrow oask$
$\tau_R \leftarrow upk^{w}$
Return $\tau_R$

$\Pi_{\mathsf{GS}}.\mathsf{Ver}_{\mathrm{GM}}^{pp}(gmsk, oapk, RL, m, \sigma)$
$(x_0,x_1,\tilde{x}_0) \leftarrow gmsk$ ; $(W,Z) \leftarrow oapk$
$(u_0,C_y,C_u,V,ct_1,ct_2,M_1,M_2,N_1,N_2,T_1,T_2,\pi) \leftarrow \sigma$
For $\tau_R \in RL$ do
    If $e(T_2/\tau_R, M_2) = e(T_1, N_2)$ then return 0
$V' \leftarrow u_0^{x_0}C_y^{x_1}/C_u$
$X_1 \leftarrow h_1^{x_1}$ ; $C_{\tilde{x}_0} \leftarrow g_1^{x_0}h_1^{\tilde{x}_0}$
$b \leftarrow \mathsf{SoK}_{\mathcal{R}_2}.\mathsf{Ver}($
    $(g_1,h_1,u_0,X_1,C_y,V',W,Z,ct_1,ct_2,M_1,M_2,N_1,N_2,T_1,T_2),m,\pi)$
Return $V == V' \wedge b$
```

$$\mathcal{R}_1 = \big\{\big((x_0,x_1,\tilde{x}_0,r),(g_1,h_1,u_0,u_1,X_1,C_{\tilde{x}_0},Y,Y_r)\big) : u_0 = g^{r} \wedge Y_r = Y^{r} \wedge u_1 = u_0^{x_0}Y_r^{x_1} \wedge C_{\tilde{x}_0} = g_1^{x_0}h_1^{\tilde{x}_0} \wedge X_1 = h_1^{x_1}\big\}$$

$$\begin{aligned}\mathcal{R}_2 = \big\{\big((y,\alpha_y,\alpha_u,\alpha_{ct},\alpha_T,r_m,r_n),(g_1,h_1,u_0,X_1,C_y,V,W,Z,ct_1,ct_2,M_1,M_2,N_1,N_2,T_1,T_2)\big) :& \\ C_y = u_0^{y}h_1^{\alpha_y} \wedge V = g_1^{-\alpha_u}X_1^{\alpha_y} \wedge ct_1 = g_1^{\alpha_{ct}} \wedge ct_2 = g_1^{y}Z^{\alpha_{ct}}& \\ \wedge M_1 = g_1^{r_m} \wedge M_2 = g_2^{r_m} \wedge N_1 = g_1^{r_n} \wedge N_2 = g_2^{r_n} \wedge T_1 = M_1^{\alpha_T} \wedge T_2 = W^{y}N_1^{\alpha_T}\big\}&\end{aligned}$$

Figure 8: Keyed-verification, multi-opener group signature with verifier-local revocation. The core primitive of Orca.

```
$\underline{\mathsf{MAC}_{\mathsf{GGM}}.\mathsf{BlindInp}^{pp_\mathsf{M}}(pk, x, oapk, m_{in} : st)}$
$(X_1, C_{\tilde{x}_0}) \leftarrow pk$ ; $(W, Z) \leftarrow oapk$
If $m_{in} == \bot$ do
    $(\gamma, r) \leftarrow\!\!\$\, \mathbb{Z}_p^2$ ; $D \leftarrow g_1^{\gamma}$
    $ct_1 \leftarrow g_1^r$ ; $ct_2 \leftarrow g_1^x D^r$
    $\hat{r} \leftarrow\!\!\$\, \mathbb{Z}_p$ ; $\hat{ct}_1 \leftarrow g_1^{\hat{r}}$ ; $\hat{ct}_2 \leftarrow g_1^x Z^{\hat{r}}$
    $\pi \leftarrow \mathsf{NiZK}_{\mathcal{R}_3}.\mathsf{Prove}((x, r, \hat{r}), (g_1, D, Z, ct_1, ct_2, \hat{ct}_1, \hat{ct}_2))$
    $m_{out} \leftarrow (D, ct_1, ct_2, \hat{ct}_1, \hat{ct}_2, \pi)$
    $st \leftarrow (\gamma, ct_1, ct_2)$
    Return $(m_{out}, \texttt{cont}, st)$
$(ct_1', ct_2', u_0, X_b, \pi) \leftarrow m_{in}$
$(\gamma, ct_1, ct_2) \leftarrow st$
$b \leftarrow \mathsf{NiZK}_{\mathcal{R}_4}.\mathsf{Ver}((g_1, h_1, X_1, X_b, C_{\tilde{x}_0}, g_1^{\gamma}, u_0, ct_1, ct_2, ct_1', ct_2'), \pi)$
If $b == 0$ then return $(\bot, \texttt{reject}, st)$
$u_1 \leftarrow ct_2' / ct_1'^{\gamma}$ ; $t \leftarrow (u_0, u_1)$ ; $st \leftarrow t$
Return $(\bot, \texttt{accept}, st)$
```

```
$\underline{\mathsf{MAC}_{\mathsf{GGM}}.\mathsf{BlindEv}^{pp_\mathsf{M}}(sk, oapk, m_{in} : st)}$
$(x_0, x_1, \tilde{x}_0) \leftarrow sk$ ; $X_1 \leftarrow h_1^{x_1}$ ; $C_{\tilde{x}_0} \leftarrow g_1^{x_0} h_1^{\tilde{x}_0}$ ; $(W, Z) \leftarrow oapk$
$(D, ct_1, ct_2, \hat{ct}_1, \hat{ct}_2, \pi) \leftarrow m_{in}$
$b \leftarrow \mathsf{NiZK}_{\mathcal{R}_3}.\mathsf{Ver}((g_1, D, Z, ct_1, ct_2, \hat{ct}_1, \hat{ct}_2), \pi)$
If $b == 0$ then return $(\bot, \texttt{reject}, st)$
$b \leftarrow\!\!\$\, \mathbb{Z}_p$ ; $r' \leftarrow\!\!\$\, \mathbb{Z}_p$ ; $b_1 \leftarrow x_1 \cdot b$
$u_0 \leftarrow g_1^b$ ; $X_b \leftarrow X_1^b$
$ct_1' \leftarrow ct_1^{b_1} g_1^{r'}$ ; $ct_2' \leftarrow ct_2^{b_1} u_0^{x_0} D^{r'}$
$\pi \leftarrow\!\!\$\, \mathsf{NiZK}_{\mathcal{R}_4}.\mathsf{Prove}((x_0, x_1, \tilde{x}_0, r', b, b_1),$
                    $(g_1, h_1, X_1, X_b, C_{\tilde{x}_0}, D, u_0, ct_1, ct_2, ct_1', ct_2'))$
$m_{out} \leftarrow (ct_1', ct_2', u_0, X_b, \pi)$
$st \leftarrow (\hat{ct}_1, \hat{ct}_2)$
Return $(m_{out}, \texttt{accept}, st)$
```

$$\mathcal{R}_3 = \left\{ \left((x, r, \hat{r}), (g_1, D, Z, ct_1, ct_2, \hat{ct}_1, \hat{ct}_2) \right) : ct_1 = g_1^r \wedge ct_2 = g_1^x D^r \wedge \hat{ct}_1 = g_1^{\hat{r}} \wedge \hat{ct}_2 = g_1^x Z^{\hat{r}} \right\}$$

$$\mathcal{R}_4 = \left\{ \left((x_0, x_1, \tilde{x}_0, r', b, b_1), (g_1, h_1, X_1, X_b, C_{\tilde{x}_0}, D, u_0, ct_1, ct_2, ct_1', ct_2') \right) : C_{\tilde{x}_0} = g_1^{x_0} h_1^{\tilde{x}_0} \wedge X_1 = h_1^{x_1} \wedge X_b = X_1^b \wedge X_b = h_1^{b_1} \wedge u_0 = g_1^b \wedge ct_1' = ct_1^{b_1} g_1^{r'} \wedge ct_2' = ct_2^{b_1} u_0^{x_0} D^{r'} \right\}$$

Figure 9: Modified blind evaluation of algebraic MACs for token generation used in the extension of Orca with one-time tokens.

adversary is given the revocation token. After interacting with these oracles, the adversary outputs a msg, user i, and opening authority j. User i signs message m to opening authority j, and the adversary wins if one of three conditions holds on the signature σ. If the signature verifies with j's revocation list, but user i was on the revocation list from REVOKE, this represents a break of correctness. The second winning condition is the opposite: if the signature does not verify, and user i is not part of the revocation list, that is also incorrect behavior. The last winning condition is if the signature opens to some value other than user i's public key.

We forgo a formal proof of correctness for our scheme, as it is relatively straightforward to confirm through inspection.

D.2 Anonymity

Anonymity is defined by the game $\mathrm{G}^{\text{anon-}b}_{\mathsf{GS},\mathcal{A}}$ shown in Figure 11. We define the advantage of adversary $\mathcal{A}$ as:

$$\mathbf{Adv}^{\text{anon}}_{\mathsf{GS},\mathcal{A}}(\lambda) = \left| \Pr\left[\mathrm{G}^{\text{anon-1}}_{\mathsf{GS},\mathcal{A}}(\lambda) = 1 \right] - \Pr\left[\mathrm{G}^{\text{anon-0}}_{\mathsf{GS},\mathcal{A}}(\lambda) = 1 \right] \right| .$$

We say that a verifier-local revocable, keyed-verification, multi-opener group signature GS is *anonymous* if $\mathbf{Adv}^{\text{anon}}_{\mathsf{GS},\mathcal{A}}(\cdot)$ is negligible for any polynomial-time adversary $\mathcal{A}$.

In the anonymity game, the adversary plays the role of the platform in attempting to determine the signer's identity of a challenge signature. The adversary may register users and opening authorities using oracles ADDU and ADDOA (denoted as ADDX for $X \in \{U, OA\}$ in the security game) and may corrupt parties to learn their secret key through oracles SKU and SKOA (denoted SKX). The adversary can generate signatures for uncorrupted users using SIGN and generate revocation tokens from honest opening authorities for arbitrary signatures using OPENREVOKE. After interacting with these oracles, the adversary may make a single challenge query to CHSIGN in which they specify two uncorrupted users i_0 and i_1 and an opening authority j and receives a signature from user i_b based on challenge bit b. To disallow trivial wins, neither user's revocation token for j can have been queried via a previous signature to OPENREVOKE prior to the challenge query, and are restricted from being queried after the challenge query. The challenge users and opening authority are also restricted from being queried to SKX following the challenge query. The adversary wins if it correctly guesses the challenge bit b.

We extend the game in $\mathrm{G}^{\text{revanon-}b}_{\mathsf{GS},\mathcal{A}}$ to capture revocation token anonymity (includes highlighted code in Figure 11). Here an additional CHREVOKE oracle is given to be run on the challenge signature to receive the revocation token for the user i_b. To prevent trivial wins where the adversary holds other signatures from the challenge signers, the CHREVOKE oracle rejects queries when either of the two challenge signing users have been queried to SIGN.

Discussion. Our anonymity definition captures an actively malicious platform with the ability to adaptively compromise signing users. Our definition captures what is commonly referred to as CCA-anonymity in the group signature literature by providing

```
$\underline{\text{Game } \mathrm{G}^{\mathrm{corr}}_{\mathsf{GS},\mathcal{A}}(\lambda)}$
$pp \leftarrow\!\!\$\ \mathsf{GS.Setup}(\lambda)$
$(gmpk, gmsk) \leftarrow\!\!\$\ \mathsf{GS.Kg}()$
$(i, j, m) \leftarrow\!\!\$\ \mathcal{A}^{\textsc{AddX},\textsc{Revoke}}(gmsk)$
If $i \notin H_U \vee j \notin H_{OA}$ then return 0
$(upk, usk) \leftarrow H_U[i]$ ; $(oapk, oask) \leftarrow H_{OA}[j]$
$\sigma \leftarrow\!\!\$\ \mathsf{GS.Sign}(usk, gmpk, oapk, m)$
$b \leftarrow \mathsf{GS.Ver}(gmsk, oapk, RL[j], m, \sigma)$
If $b = 1 \wedge i \in RL_U[j]$ then return 1
If $b = 0 \wedge i \notin RL_U[j]$ then return 1
$upk' \leftarrow\!\!\$\ \mathsf{GS.Open}(oask, gmpk, m, \sigma)$
If $upk \neq upk'$ then return 1
Return 0
```

```
$\underline{\textsc{AddX}(i)_{X \in \{U, OA\}}}$
Require $i \notin H_X$
$st \leftarrow \perp$ ; $st_{GM} \leftarrow \perp$
$m_{in} \leftarrow \perp$ ; $dec \leftarrow \texttt{cont}$
While $dec = \texttt{cont}$ do
    $(m_{in}, dec) \leftarrow\!\!\$\ \mathsf{GS.JoinX}(gmpk, m_{in} : st)$
    $(m_{in}, dec) \leftarrow\!\!\$\ \mathsf{GS.IssueX}(gmsk, m_{in} : st_{GM})$
If $dec = \texttt{accept}$ then
    $(pk, sk) \leftarrow st$
    $REG_X[i] \leftarrow pk$ ; $H_X[i] \leftarrow (pk, sk)$

$\underline{\textsc{Revoke}(i, j)}$
Require $i \in H_U \wedge j \in H_{OA}$
$(upk, usk) \leftarrow H_U[i]$ ; $(oapk, oask) \leftarrow H_{OA}[j]$
$\tau_R \leftarrow\!\!\$\ \mathsf{GS.Revoke}(oask, upk)$
$RL[j] \hookleftarrow \tau_R$ ; $RL_U[j] \hookleftarrow i$
Return $\tau_R$
```

Figure 10: Correctness game for keyed-verification multi-opener group signatures.

access to an opening oracle $\textsc{OpenRevoke}$, allowing the adversary to maul signatures in an arbitrary fashion. This treatment is slightly different from previous formalizations that do not handle verifier-local revocation in that we choose not to explicitly provide a separate "open" oracle, but rather combine the open and revoke functionality into a single oracle. This choice fits the setting where the open algorithm is run locally by the recipient and only the revocation token is ever sent to the platform. Nevertheless, adding a separate "open" oracle would not affect the security of our scheme; it is omitted for simplification.

We also consider rogue key attacks; we allow the adversary to arbitrarily create public keys for corrupted parties, i.e., the group manager and opening authorities, but require the adversary to prove knowledge of secret keys. We model this, for simplicity, by asking the adversary to produce a valid secret key for a public key during key generation, following the knowledge of secret key (KOSK) model of [11]. This model can in turn be instantiated by including proofs of knowledge of secret keys that we can extract from to proceed with the proof in the KOSK model. In the game pseudocode, we use a wellformed predicate to capture this check, i.e., for a discrete log public key X and secret key x, checks $X = g^x$.

Many previous schemes that achieved verifier-local revocation targeted a weaker form of anonymity called *selfless anonymity* meaning that user signatures can be deanonymized by their own secret key. Our scheme also targets selfless anonymity, as shown by the query restriction on $\textsc{SKU}$, however it seems possible that the keyed-verification setting may allow for an efficient scheme with full anonymity and verifier-local revocation; we leave this to future work.

Lastly, in addition to anonymity of the signature, we also target anonymity of the revocation token. To our knowledge, our extension of the anonymity game to capture anonymity of the revocation token is the first definitional attempt at doing so. Revocation anonymity implies the anonymity as the security games are equivalent if the challenge revoke oracle ($\textsc{ChRevoke}$) of the revocation anonymity game is not called.

Theorem 1. *Let Π_{GS} be the keyed-verification, multi-opener group signature scheme defined in Figure 8 over prime order p cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$. Let $\mathsf{MAC}_{\mathsf{GGM}}$ be the keyed-verification anonymous credentials scheme from [22] on $\mathbb{G}_1$. Let ElG be the ElGamal encryption scheme on $\mathbb{G}_1$. Then for any adversary $\mathcal{A}$ against the anonymity of Π_{GS}, we give adversaries $\mathcal{A}_1$ to $\mathcal{A}_6$ such that*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{anon}}_{\Pi_{\mathsf{GS}},\mathcal{A}}(\lambda) \leq 2q_u^2 q_{oa}\big(&\mathbf{Adv}^{\mathrm{sound}}_{\mathsf{NiZK}_{\mathcal{R}_1},\mathcal{A}_1,\mathcal{X}_{\mathcal{R}_1}}(\lambda) + \mathbf{Adv}^{\mathrm{anon}}_{\mathsf{MAC}_{\mathsf{GGM}},\mathcal{A}_2,\mathcal{S}_{\mathsf{MAC}}}(\lambda) + \mathbf{Adv}^{\mathrm{simext}}_{\mathsf{SoK}_{\mathcal{R}_2},\mathcal{A}_3,\mathcal{X}_{\mathcal{R}_2},\mathcal{S}_{\mathcal{R}_2}}(\lambda) \\ &+ \mathbf{Adv}^{\mathrm{indcpa}}_{\mathsf{ElG},\mathcal{A}_4}(\lambda) + 2\cdot\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G}_1,p,\mathcal{A}_5}(\lambda) + \mathbf{Adv}^{\mathrm{dlin}}_{\mathbb{G}_1,\mathbb{G}_2,p,\mathcal{A}_6}(\lambda)\big)\end{aligned}$$

where $\mathcal{A}$ makes at most q_u and q_{oa} queries to the add user and add opening authority oracles, respectively.

Proof. We bound the advantage of $\mathcal{A}$ by bounding the advantage of each of a series of game hops. We define $\mathrm{G}^b = \mathrm{G}^{\mathrm{anon}\text{-}b}_{\Pi_{\mathsf{GS}},\mathcal{A}}(\lambda)$ and define games $\mathrm{G}^b_{\mathrm{A}}$, $\mathrm{G}^b_{\mathrm{B}}$, $\mathrm{G}^b_{\mathrm{C}}$, $\mathrm{G}^b_{\mathrm{D}}$, $\mathrm{G}^b_{\mathrm{E}}$, $\mathrm{G}^b_{\mathrm{F0}}$, $\mathrm{G}^b_{\mathrm{F1}}$, G_{G} to gradually transform the view of the adversary until in G_{G} it is no longer dependent on bit b. The inequality above follows from simple calculations based on the following claims which we will justify:

```
Game $\mathrm{G}^{\text{anon-}b}_{\mathsf{GS},\mathcal{A}}(\lambda)$ ; $\mathrm{G}^{\text{revanon-}b}_{\mathsf{GS},\mathcal{A}}(\lambda)$
$pp \leftarrow\!\!\$\ \mathsf{GS.Setup}(\lambda)$
$(gmpk, gmsk) \leftarrow\!\!\$\ \mathcal{A}(\bot : st_{\mathcal{A}})$
Require $\mathsf{GS.wellformed}_{\mathrm{GM}}(\lambda, gmpk, gmsk)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{WRegOA},\textsc{Sign},\textsc{ChSign},\textsc{AddX},\textsc{SKU},\textsc{OpenRevoke},\ \textsc{ChRevoke}}(\bot : st_{\mathcal{A}})$
Return $b'$

$\textsc{WRegOA}(i, pk, sk)$
Require $\mathsf{GS.wellformed}_{\mathrm{OA}}(\lambda, pk, sk)$
If $i \in H_{OA}$ then $(pk, sk) \leftarrow H_{OA}$
$REG_{OA}[i] \leftarrow (pk, sk)$

$\textsc{Sign}(i, j, m)$
Require $i \in H_U \wedge j \in REG_{OA}$
$(upk, usk) \leftarrow H_U[i]$ ; $(oapk, oask) \leftarrow REG_{OA}[j]$
$\sigma \leftarrow\!\!\$\ \mathsf{GS.Sign}(usk, gmpk, oapk, m)$
$\Sigma[j][\sigma] \leftarrow i$
Return $\sigma$

$\textsc{ChSign}(i_0, i_1, j, m)$
Require $i_0, i_1 \notin K_U \wedge j \notin K_{OA}$
Require $i_0, i_1 \in H_U \wedge j \in H_{OA}$
Require $i_0, i_1 \notin RL[j]$
$(upk_0, usk_0) \leftarrow H_U[i_0]$ ; $(upk_1, usk_1) \leftarrow H_U[i_1]$
$(oapk, oask) \leftarrow H_{OA}[j]$
$\sigma \leftarrow\!\!\$\ \mathsf{GS.Sign}(usk_b, gmpk, oapk, m)$
$RQ[j] \hookleftarrow [i_0, i_1]$
$\tilde{\Sigma}[\sigma] \leftarrow (i_0, i_1, j)$
Return $\sigma$
```

```
$\textsc{AddX}(i, m_{in})_{X \in \{U, OA\}}$
Require $i \notin H_X$
$(m_{in}, dec) \leftarrow\!\!\$\ \mathsf{GS.JoinX}(gmpk, m_{in} : st_X[i])$
If $dec = \mathtt{accept}$ then
    $(pk, sk) \leftarrow st_X[i]$ ; $H_X[i] \leftarrow (pk, sk)$
Return $(m_{in}, dec)$

$\mathrm{SKX}(i)_{X \in \{U, OA\}}$
Require $i \notin RQ[*] \wedge j \notin RQ$
$K_X \hookleftarrow i$
Return $H_X[i]$

$\textsc{OpenRevoke}(m, \sigma, j)$
Require $j \in H_{OA}$
Require $\sigma \notin \tilde{\Sigma}$
Require $\sigma \notin \Sigma[j] \vee \Sigma[j][\sigma] \notin RQ[j]$
$(oapk, oask) \leftarrow H_{OA}[j]$
$upk \leftarrow \mathsf{GS.Open}(oask, gmpk, m, \sigma)$
$\tau_R \leftarrow\!\!\$\ \mathsf{GS.Revoke}(oask, upk)$
If $\sigma \in \Sigma[j]$ do $RL[j] \hookleftarrow \Sigma[j][\sigma]$
Return $\tau_R$

$\textsc{ChRevoke}(\sigma)$
Require $\sigma \in \tilde{\Sigma}$ ; $(i_0, i_1, j) \leftarrow \tilde{\Sigma}[\sigma]$
Require $i_0, i_1 \notin \Sigma[*][*]$
$(upk, usk) \leftarrow H_U[i_b]$ ; $(oapk, oask) \leftarrow H_{OA}[j]$
$\tau_R \leftarrow\!\!\$\ \mathsf{GS.Revoke}(oask, upk)$
Return $\tau_R$
```

Figure 11: Anonymity game for keyed-verification multi-opener group signatures. An extension to the anonymity game is provided to capture anonymity of revocation tokens which includes the highlighted code.

(1) $\mathbf{Adv}^{\text{anon}}_{\Pi_{\mathsf{GS}},\mathcal{A}}(\lambda) = \left|\Pr[\mathrm{G}^0 = 1] - \Pr[\mathrm{G}^1 = 1]\right| \leq q_u^2 q_{oa} \cdot \left|\Pr[\mathrm{G}^0_{\mathrm{A}} = 1] - \Pr[\mathrm{G}^1_{\mathrm{A}} = 1]\right|$

(2) $|\Pr[\mathrm{G}^b_{\mathrm{A}} = 1] - \Pr[\mathrm{G}^b_{\mathrm{B}} = 1]| = \mathbf{Adv}^{\text{sound}}_{\mathsf{NiZK}_{\mathcal{R}_1},\mathcal{A}_1,\mathcal{X}_{\mathcal{R}_1}}(\lambda)$

(3) $|\Pr[\mathrm{G}^b_{\mathrm{B}} = 1] - \Pr[\mathrm{G}^b_{\mathrm{C}} = 1]| = \mathbf{Adv}^{\text{anon}}_{\mathsf{MAC}_{\mathsf{GGM}},\mathcal{A}_2,\mathcal{S}_{\mathsf{MAC}}}(\lambda)$

(4) $|\Pr[\mathrm{G}^b_{\mathrm{C}} = 1] - \Pr[\mathrm{G}^b_{\mathrm{D}} = 1]| = \mathbf{Adv}^{\text{simext}}_{\mathsf{SoK}_{\mathcal{R}_2},\mathcal{A}_3,\mathcal{X}_{\mathcal{R}_2},\mathcal{S}_{\mathcal{R}_2}}(\lambda)$

(5) $|\Pr[\mathrm{G}^b_{\mathrm{D}} = 1] - \Pr[\mathrm{G}^b_{\mathrm{E}} = 1]| = \mathbf{Adv}^{\text{indcpa}}_{\mathsf{ElG},\mathcal{A}_4}(\lambda)$

(6) $|\Pr[\mathrm{G}^b_{\mathrm{E}} = 1] - \Pr[\mathrm{G}^b_{\mathrm{F1}} = 1]| = 2 \cdot \mathbf{Adv}^{\text{ddh}}_{\mathbb{G}_1,p,\mathcal{A}_5}(\lambda)$

(7) $|\Pr[\mathrm{G}^b_{\mathrm{F1}} = 1] - \Pr[\mathrm{G}_{\mathrm{G}} = 1]| = \mathbf{Adv}^{\text{dlin}}_{\mathbb{G}_1,\mathbb{G}_2,p,\mathcal{A}_7}(\lambda)$

Sketch: Recall the group signature is composed of three components: (i) the identity ciphertext ct_{id} enclosing the signer's public key to the opening authority, (ii) the revocation ciphertext ct_R enclosing the revocation token, and (iii) a zero knowledge proof π that (i) and (ii) were constructed properly with knowledge of a key pair (y, Y) and a MAC t on Y. To remove the dependence of signing on challenge bit b, our proof steps through each of these components in sequence. Claims 2 and 3 remove the use of signing key y_b in creating (iii) the zero knowledge proof π of a valid MAC. Claims 4 and 5 remove the use of signing key y_b in encrypting (i) the identity ciphertext. And lastly, claims 6 and 7 remove the use of signing key y_b in constructing (ii) the revocation ciphertext.

Claim 1: Without loss of generality, assume calls to $\textsc{AddX}$ are made with incrementing indices, e.g., $i = 1, 2, \ldots, q$. $\mathrm{G}^b_{\mathrm{A}}$ is the same as G^b except it guesses the parties i_0, i_1, j on which $\mathcal{A}$ will make its $\textsc{ChSign}$ query and aborts if it is incorrect. If $\mathcal{A}$ makes its $\textsc{ChSign}$ query on a different set of parties, if it queries $\textsc{OpenRevoke}$ with a $\textsc{Sign}$ signature from i_0 or i_1, or calls SKX on any of i_0, i_1, j, then $\mathrm{G}^b_{\mathrm{A}}$ sets a $\mathsf{bad}^b_{\mathrm{A}}$ flag and aborts. By an identical-until-bad argument and the fundamental lemma of game playing [8], we have that

$$\Pr[\mathrm{G}^b = 1 \land \neg \mathsf{bad}_{\mathrm{A}}^b] = \Pr[\mathrm{G}_{\mathrm{A}}^b = 1 \land \neg \mathsf{bad}_{\mathrm{A}}^b] .$$

And since $\mathrm{G}_{\mathrm{A}}^b$ aborts and outputs 0 when $\mathsf{bad}_{\mathrm{A}}^b$ is set, i.e., only outputs 1 when $\neg\mathsf{bad}_{\mathrm{A}}^b$, we have

$$\begin{aligned}\Pr[\mathrm{G}_{\mathrm{A}}^b = 1] &= \Pr[\mathrm{G}_{\mathrm{A}}^b = 1 | \neg \mathsf{bad}_{\mathrm{A}}^b] \Pr[\neg \mathsf{bad}_{\mathrm{A}}^b] + \Pr[\mathrm{G}_{\mathrm{A}}^b = 1 | \mathsf{bad}_{\mathrm{A}}^b] \Pr[\mathsf{bad}_{\mathrm{A}}^b] \\ &= \Pr[\mathrm{G}_{\mathrm{A}}^b = 1 | \neg \mathsf{bad}_{\mathrm{A}}^b] \Pr[\neg \mathsf{bad}_{\mathrm{A}}^b] \\ &= \Pr[\mathrm{G}_{\mathrm{A}}^b = 1 \land \neg \mathsf{bad}_{\mathrm{A}}^b] .\end{aligned}$$

Then, we have

$$\begin{aligned}\left|\Pr[\mathrm{G}_{\mathrm{A}}^0 = 1] - \Pr[\mathrm{G}_{\mathrm{A}}^1 = 1]\right| &= \left|\Pr[\mathrm{G}^0 = 1 \land \neg \mathsf{bad}_{\mathrm{A}}^b] - \Pr[\mathrm{G}^1 = 1 \land \neg \mathsf{bad}_{\mathrm{A}}^b]\right| \\ &= \left|\Pr[\mathrm{G}^0 = 1] \cdot \Pr[\neg \mathsf{bad}_{\mathrm{A}}^b] - \Pr[\mathrm{G}^1 = 1] \cdot \Pr[\neg \mathsf{bad}_{\mathrm{A}}^b]\right| \qquad (1) \\ &= \Pr[\neg \mathsf{bad}_{\mathrm{A}}^0] \cdot \left|\Pr[\mathrm{G}^0 = 1] - \Pr[\mathrm{G}^1 = 1]\right| , \qquad (2)\end{aligned}$$

where (1) holds because the condition to set $\mathsf{bad}_{\mathrm{A}}^b$ is independent of the rest of the game G^b, and (2) holds since $\Pr[\neg\mathsf{bad}_{\mathrm{A}}^0] = \Pr[\neg\mathsf{bad}_{\mathrm{A}}^b]$; the probability the guess is correct is independent of bit b.

Lastly since the parties are guessed at random, the probability that the guess is correct and bad_b is not set is at least

$$\Pr[\neg \mathsf{bad}_{\mathrm{A}}^0] \geq \frac{1}{q_u^2 q_{oa}} .$$

Claim 2: The next game $\mathrm{G}_{\mathrm{B}}^b$ checks the knowledge soundness of the issuance proofs for honest group members that accept the interactive join protocol in ADDU by using the extractor for $\mathsf{NiZK}_{\mathcal{R}_1}$ to extract and double check proofs (line 15 in Figure 8). If the extractor fails, a $\mathsf{bad}_{\mathrm{B}}^b$ flag is set and the game is aborted. By an identical-until-bad argument via the fundamental lemma of game playing [8],

$$|\Pr[\mathrm{G}_{\mathrm{A}}^b = 1] - \Pr[\mathrm{G}_{\mathrm{B}}^b = 1]| \leq \Pr[\mathsf{bad}_{\mathrm{B}}^b] .$$

We bound the probability $\mathsf{bad}_{\mathrm{B}}^b$ is set exactly by the advantage against the knowledge soundness of $\mathsf{NiZK}_{\mathcal{R}_1}$, constructing an adversary $\mathcal{A}_1^b$ that wins the game whenever $\mathsf{bad}_{\mathrm{B}}^b$ is set by returning the proof and statement that failed extraction. We construct $\mathcal{A}_1$ by running $\mathcal{A}_1^0$ and $\mathcal{A}_1^1$ each with probability 1/2.

Claim 3: The previous game ensures that the credentials issued to honest users are properly generated. The next game $\mathrm{G}_{\mathrm{C}}^b$ replaces the construction of proof π and values u_0, C_y, C_u, V in SIGN and CHSIGN (line 58, 62-64 of Figure 8) with the output of a simulator. We observe that the generation of π through $\mathcal{R}_2$ is exactly that of showing a credential in $\mathsf{MAC}_{\mathsf{GGM}}$ [22] with the added relation on MAC value y:

$$\phi(y) = ct_1 = g_1^{\alpha_{ct}} \land ct_2 = g_1^y Z^{\alpha_{ct}} \land \, M_1 = g_1^{r_m} \land M_2 = g_2^{r_m} \land N_1 = g_1^{r_n} \land N_2 = g_2^{r_n} \land \, T_1 = M_1^{\alpha_T} \land T_2 = W^y N_1^{\alpha_T} .$$

By the KVAC anonymity of $\mathsf{MAC}_{\mathsf{GGM}}$ [22, Definition 7], we have a simulator $\mathcal{S}_{\mathsf{MAC}}$ that can simulate the output of credential showing, (π, u_0', C_y, C_u, V), given only the secret key of the issuer, $gmsk$, without values of the MAC, y. Under the hood, the KVAC simulator $\mathcal{S}_{\mathsf{MAC}}$ is constructed using the signature of knowledge simulator $\mathcal{S}_{\mathcal{R}_2}$ to simulate π with special constructions of u_0', C_y, C_u, V using $gmsk$. We provide the pseudocode for $\mathcal{S}_{\mathsf{MAC}}$ from [22, Theorem 4] that makes use of $\mathcal{S}_{\mathcal{R}_2}$. This will be important since in a later game hop, we will make use of simulation extractability over simulated proofs from $\mathcal{S}_{\mathcal{R}_2}$.

```
Simulator $\mathcal{S}_{\mathsf{MAC}}(\lambda)$
$\mathcal{S}_{\mathsf{MAC}}.\mathsf{Prove}^{pp_{\mathsf{M}}}(gmsk, gmpk, (W, Z, ct_1, ct_2, M_1, M_2, N_1, N_2, T_1, T_2), m)$
$(p, \mathbb{G}_1, g_1, h_1) \leftarrow pp_{\mathsf{M}}$
$(x_0, x_1, \tilde{x}_0) \leftarrow gmsk$ ; $(X_1, C_{\tilde{x}_0}) \leftarrow gmpk$
$(u_0, C_y, C_u) \leftarrow\!\!\$\, \mathbb{G}_1^3$
$V \leftarrow u_0^{x_0} C_y^{x_1} / C_u$
$\pi \leftarrow\!\!\$\, \mathcal{S}_{\mathcal{R}_2}.\mathsf{Prove}(\xi, (g_1, h_1, u_0, X_1, C_y, V, W, Z, ct_1, ct_2, M_1, M_2, N_1, N_2, T_1, T_2), m)$
Return $(\pi, u_0, C_y, C_u, V)$
```

We can bound the distinguishing advantage between $\mathrm{G}_{\mathrm{B}}^{b}$ and $\mathrm{G}_{\mathrm{C}}^{b}$ exactly by the advantage against the KVAC anonymity game, constructing an adversary $\mathcal{A}_2^b$ that simulates $\textsc{Sign}$ and $\textsc{ChSign}$ for $\mathcal{A}$ by generating the proof through their own show oracle. We construct $\mathcal{A}_2$ from $\mathcal{A}_2^0$ and $\mathcal{A}_2^1$ analogously to before.

Claim 4: Our goal in the next game hop (to $\mathrm{G}_{\mathrm{D}}^{b}$) is to respond to $\textsc{OpenRevoke}$ queries without using $oask$. This is done using the simulation extractability extractor for $\mathsf{SoK}_{\mathcal{R}_2}$. Recall in the previous game, all signatures created by honest parties are simulated using the simulator $\mathcal{S}_{\mathcal{R}_2}$. In game $\mathrm{G}_{\mathrm{D}}^{b}$, these simulated signatures are tracked in a table along with the signing user and intended opening authority.

If a simulated signature is passed to $\textsc{OpenRevoke}$ for opening authority, the oracle responds using the table. More specifically, the intended opening authority is looked up and if it does not match, then $\perp$ is returned. Otherwise, the signing key y for the signing user is looked up and the revocation token is calculated as $\tau_R \leftarrow oapk_j^y$. By the correctness property of the group signature, this matches the behavior of $\mathsf{GS.Open}$ since a signature for a different opening authority will not open and looking up the signing user upk simulates exactly running $\mathsf{GS.Open}$ for valid signatures.

If a non-simulated signature is passed to $\textsc{OpenRevoke}$ and the proof π verifies under $oapk$, the extractor for $\mathsf{SoK}_{\mathcal{R}_2}$ is run to extract y and the revocation token is calculated as $\tau_R \leftarrow oapk_j^y$. If the extractor fails, a $\mathsf{bad}_{\mathrm{D}}^{b}$ flag is set and the game is aborted. By an identical-until-bad argument via the fundamental lemma of game playing [8],

$$|\Pr[\mathrm{G}_{\mathrm{C}}^{b} = 1] - \Pr[\mathrm{G}_{\mathrm{D}}^{b} = 1]| \leq \Pr[\mathsf{bad}_{\mathrm{D}}^{b}] \,.$$

We bound the probability $\mathsf{bad}_{\mathrm{D}}^{b}$ is set exactly by the advantage against the simulation extractability of $\mathsf{NiZK}_{\mathcal{R}_2}$, constructing an adversary $\mathcal{A}_1^b$ that simulates signing proofs using its $\textsc{SimProve}$ oracle and wins the game whenever $\mathsf{bad}_{\mathrm{D}}^{b}$ is set by returning the proof and statement that failed extraction. We construct $\mathcal{A}_3$ from $\mathcal{A}_3^0$ and $\mathcal{A}_3^1$ analogously to before.

Claim 5: $\mathrm{G}_{\mathrm{E}}^{b}$ replaces the ElGamal encryption of $Y_b = g_1^{y_b}$ in $\textsc{ChSign}$ (line 60 of Figure 8) with an encryption of g_1, a group element independent of bit b. We bound the distinguishing advantage between $\mathrm{G}_{\mathrm{D}}^{b}$ and $\mathrm{G}_{\mathrm{E}}^{b}$ by the IND-CPA security of ElG which in turn is dependent on DDH in $\mathbb{G}_1$ (external Diffie-Hellman assumption). We construct IND-CPA adversary $\mathcal{A}_4^b$ that sets the encryption key of opening authority j (Z_j) to the key output from the IND-CPA game. $\mathcal{A}_4^b$ simulates $\textsc{ChSign}$ generating (ct_1, ct_2) by passing $(g_1^{y_b}, g_1)$ to its left-right encryption oracle. Due to $\mathrm{G}_{\mathrm{D}}^{b}$, the decryption of ct in $\textsc{OpenRevoke}$ is not run, and instead the table of simulated signatures or extractor is used to respond to queries — both of which are done without the use of the decryption key of $oask$. Thus, $\mathcal{A}_4^b$ exactly runs $\mathrm{G}_{\mathrm{D}}^{b}$ and $\mathrm{G}_{\mathrm{E}}^{b}$. We construct $\mathcal{A}_4$ from $\mathcal{A}_4^0$ and $\mathcal{A}_4^1$ analogously to before.

Claim 6: $\mathrm{G}_{\mathrm{F0}}^{b}$ replaces the revocation token of opening authority j for user i_0 ($\tau_{i_0,j} = W_j^{y_0}$) with a random group element (line 59 of Figure 8). We bound the distinguishing advantage between $\mathrm{G}_{\mathrm{E}}^{b}$ and $\mathrm{G}_{\mathrm{F0}}^{b}$ by DDH. We construct an adversary $\mathcal{A}_{5,0}^b$ for the DDH game that takes input $(A = g^\alpha, B = g^\beta, C)$ and assigns A as the public key of i_0, assigns B as the public key of j, and sets C as j's revocation token for i_0.

To construct a DDH adversary, we must ensure that we can simulate all the oracles without knowledge of i_0's signing key y_0 or j's revocation key w. From $\mathrm{G}_{\mathrm{A}}^{b}$, $\mathcal{A}_{6,0}^b$ aborts when SKX is called i_0 or j, so y_0 or w are never exposed. From $\mathrm{G}_{\mathrm{C}}^{b}$, $\textsc{Sign}/\textsc{ChSign}$ are simulated with $\mathcal{S}_{\mathcal{R}_2}$. Generating the identity ciphertext does not need knowledge of y_0. Generating the revocation ciphertext uses y_0 to calculate the revocation token, but since we are in the KOSK setting, we can instead use the secret key of the target opening authority w', and calculate $\tau \leftarrow A^{w'}$. The only opening authority for which we do not have the secret key is j, where we set the revocation token to C.

Lastly, from $\mathrm{G}_{\mathrm{D}}^{b}$, simulating the revocation token in $\textsc{OpenRevoke}$ uses the table of simulated signatures and the extractor, and importantly does not need the revocation secret key w. If a simulated signature for i_0 is passed to $\textsc{OpenRevoke}$, it will return $\perp$. Thus, $\mathcal{A}_{5,0}^b$ runs $\mathrm{G}_{\mathrm{E}}^{b}$ and $\mathrm{G}_{\mathrm{F0}}^{b}$ exactly.

We next define $\mathrm{G}_{\mathrm{F1}}^{b}$ and $\mathcal{A}_{5,1}^b$ analogously to $\mathrm{G}_{\mathrm{F0}}^{b}$ and $\mathcal{A}_{5,0}^b$ where $\mathrm{G}_{\mathrm{F1}}^{b}$ is the same as $\mathrm{G}_{\mathrm{F0}}^{b}$ except it replaces j's revocation token for user i_1 with a random group element. We construct $\mathcal{A}_5$ from $\mathcal{A}_{5,0}^0, \mathcal{A}_{5,1}^0, \mathcal{A}_{5,0}^1, \mathcal{A}_{5,1}^1$ analogously as before.

Claim 7: $\mathrm{G}_{\mathrm{G}}^{b}$ is the same $\mathrm{G}_{\mathrm{F1}}^{b}$ except that instead of sampling two separate random values for $\tau_{i_0,j}$ and $\tau_{i_1,j}$, it samples a single shared random value $R = \tau_{i_0,j} = \tau_{i_1,j}$. Observe that with this change, the output $\textsc{ChSign}$ is now independent of b: (1) the proof is simulated by $\mathcal{S}_{\mathcal{R}_2}$, (2) the ciphertext (ct_1, ct_2) enclose g_1, and (3) the revocation ciphertext (T_1, T_2) encloses a shared revocation token R. Thus $\mathrm{G}_{\mathrm{G}}^{0} = \mathrm{G}_{\mathrm{G}}^{1}$ and we call this game G_{G}.

We bound the distinguishing advantage of $\mathrm{G}_{\mathrm{F1}}^{b}$ and G_{G} by the advantage in the DLIN game using a trick to embed the DLIN elements into $\tau_{i_0,j}$ and $\tau_{i_1,j}$ [15]. Define $\mathcal{A}_6^b$ that takes DLIN input $(m_1, n_1, l_1, m_1^a, n_1^b, l_1', m_2, n_2, l_2)$, samples random group element R and sets $\tau_{i_0,j} = l_1' R / l_1^a$ and $\tau_{i_1,j} = R l_1^b$. Note that since a and b are unknown, $\tau_{i_0,j}$ and $\tau_{i_1,j}$ can not be explicitly calculated, but we show how they can still be enclosed in T_1, T_2. Second, note that when $l'_1 = l^c_1$, $\tau_{i_0,j}$ and $\tau_{i_1,j}$ are different independent random values, but when $l'_1 = l^{a+b}_1$, they are equal to the same random value,

$$\tau_{i_0,j} = \frac{l'_1 R}{l^a_1} = \frac{l^{a+b}_1 R}{l^a_1} = l^b_1 R = \tau_{i_1,j}\,.$$

Even without being able to calculate the revocation tokens directly, $\mathcal{A}^b_6$ can still simulate $\textsc{ChSign}$ and construct T_1, T_2 as follows. $\mathcal{A}^b_6$ samples $r, s, t \leftarrow\!\!\$\ \mathbb{Z}_p$.

- $\mathcal{A}^0_6$ encloses $\tau_{i_0,j}$ by constructing

$$T_1 \leftarrow m^a_1 m^s_1 \qquad T_2 \leftarrow l'_1 l^s_1 (m^a_1)^t m^{st}_1 R$$
$$M_1 \leftarrow m^r_1 \qquad M_2 \leftarrow m^r_2 \qquad N_1 \leftarrow (l_1 m^t_1)^r \qquad N_2 \leftarrow (l_2 m^t_2)^r$$

  Let $\alpha = (a+s)/r$, then $T_1 = M^\alpha_1$ and $T_2 = \tau_{i_0,j} N^\alpha_1$.

- $\mathcal{A}^1_6$ encloses $\tau_{i_1,j}$ by constructing

$$T_1 \leftarrow n^b_1 n^s_1 \qquad T_2 \leftarrow \frac{(n^b_1)^t n^{st}_1 R}{l^s_1}$$
$$M_1 \leftarrow n^r_1 \qquad M_2 \leftarrow n^r_2 \qquad N_1 \leftarrow \left(\frac{n^t_1}{l_1}\right)^r \qquad N_2 \leftarrow \left(\frac{n^t_2}{l_2}\right)^r$$

  Let $\alpha = (b+s)/r$, then $T_1 = M^\alpha_1$ and $T_2 = \tau_{i_1,j} N^\alpha_1$.

In both cases, (T_1, T_2) are constructed with some random M_1, N_1 and random α, so $\mathcal{A}^b_6$ perfectly simulates the $\textsc{ChSign}$ oracle. We construct $\mathcal{A}_6$ from $\mathcal{A}^0_6, \mathcal{A}^1_6$ analogously to as before. □

Lemma 1. *Let Π_{GS} be the keyed-verification, multi-opener group signature scheme defined in Figure 8 over prime order p cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$. Let $\mathsf{MAC}_{\mathsf{GGM}}$ be the keyed-verification anonymous credentials scheme from [22] on $\mathbb{G}_1$. Let ElG be the ElGamal encryption scheme on $\mathbb{G}_1$. Then for any adversary $\mathcal{A}$ against the anonymity of Π_{GS}, we give adversaries $\mathcal{A}_1$ to $\mathcal{A}_6$ such that*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{revanon}}_{\Pi_{\mathsf{GS}},\mathcal{A}}(\lambda) \le 2q_u^2 q_{oa}\big(&\mathbf{Adv}^{\mathsf{sound}}_{\mathsf{NiZK}_{\mathcal{R}_1},\mathcal{A}_1,\mathcal{X}_{\mathcal{R}_1}}(\lambda) + \mathbf{Adv}^{\mathsf{anon}}_{\mathsf{MAC}_{\mathsf{GGM}},\mathcal{A}_2,\mathcal{S}_{\mathsf{MAC}}}(\lambda) + \mathbf{Adv}^{\mathsf{simext}}_{\mathsf{SoK}_{\mathcal{R}_2},\mathcal{A}_3,\mathcal{X}_{\mathcal{R}_2},\mathcal{S}_{\mathcal{R}_2}}(\lambda)\\ &+ \mathbf{Adv}^{\mathsf{indcpa}}_{\mathsf{ElG},\mathcal{A}_4}(\lambda) + 2\cdot\mathbf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1,p,\mathcal{A}_6}(\lambda) + \mathbf{Adv}^{\mathsf{dlin}}_{\mathbb{G}_1,\mathbb{G}_2,p,\mathcal{A}_6}(\lambda)\big)\end{aligned}$$

where $\mathcal{A}$ makes at most q_u and q_{oa} queries to the add user and add opening authority oracles, respectively.

Proof. The proof follows exactly the same game hops as the group signature anonymity proof. In $\mathrm{G}^b_{\mathrm{D}}$ and $\mathrm{G}^b_{\mathrm{E}}$, $\textsc{ChRevoke}$ is simulated using the identity mappings in the table of simulated signatures, same as in $\textsc{OpenRevoke}$. In $\mathrm{G}^b_{\mathrm{F0}}$, $\mathrm{G}^b_{\mathrm{F1}}$, G_{G}, $\textsc{ChRevoke}$ is simulated by responding with the random group element chosen as the revocation token for i_b. G_{G} again has no dependence on the challenge bit b. □

D.3 Traceability

Traceability is defined by the game $\mathrm{G}^{\mathsf{trace}}_{\mathsf{GS},\mathcal{A}}$ shown in Figure 12. We define the advantage of adversary $\mathcal{A}$ as:

$$\mathbf{Adv}^{\mathsf{trace}}_{\mathsf{GS},\mathcal{A}}(\lambda) = \Pr\big[\mathrm{G}^{\mathsf{trace}}_{\mathsf{GS},\mathcal{A}}(\lambda) = 1\big]$$

We say that a verifier-local revocable, keyed-verification, multi-opener group signature GS is *traceable* if $\mathbf{Adv}^{\mathsf{trace}}_{\mathsf{GS},\mathcal{A}}(\cdot)$ is negligible for any polynomial-time adversary $\mathcal{A}$.

In the traceability game, the adversary plays the role of a set of malicious users and opening authorities with the goal of creating a message, signature pair that verifies under the honest platform, but fails to open at the recipient. The adversary may register as users and opening authorities using $\textsc{AddX}$. The adversary may verify arbitrary signatures under arbitrary revocation lists using $\textsc{Verify}$. Note that a verify oracle is necessary for the keyed-verification setting. After interacting with these oracles, the adversary outputs a message, signature pair along with a revocation list. The adversary wins if the signature verifies, and the open algorithm fails by either returning $\perp$ or returning an unregistered public key upk.

Discussion. The traceability game necessarily considers an honest platform, since it is trivial for the platform to issue unregistered credentials and win the game. The non-frameability game addresses the forging ability of a malicious platform.

Again we prove security in the knowledge of secret key model [11], where we add a $\textsc{KoskX}$ oracle to complete registration by providing a wellformed secret key after a public key was accepted by $\textsc{AddX}$. As before, instantiating the scheme with proofs of knowledge of secret keys during registration allows the proof to proceed as in the KOSK model.

```
$\underline{\text{Game } \mathrm{G}^{\text{trace}}_{\mathsf{GS},\mathcal{A}}(\lambda)}$
$pp \leftarrow\!\!\$\ \mathsf{GS.Setup}(\lambda)$
$(gmpk, gmsk) \leftarrow\!\!\$\ \mathsf{GS.Kg}()$
$(j, m, \sigma, L) \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Verify},\textsc{AddX}}(gmpk)$
Assert $j \in REG_{OA}$ ; $(oapk, oask) \leftarrow REG_{OA}[j]$
$b_{ver} \leftarrow \mathsf{GS.Ver}(gmsk, oapk, L, m, \sigma)$
$upk \leftarrow \mathsf{GS.Open}(oask, gmpk, m, \sigma)$
$b_{opn1} \leftarrow upk == \perp \vee upk \notin REG_U$
Return $b_{ver} \wedge b_{opn1}$

$\underline{\textsc{KoskX}(i, sk)_{X \in \{U,OA\}}}$
$P_X[i] \leftarrow sk$
```

```
$\underline{\textsc{AddX}(i, m_{in})_{X \in \{U,OA\}}}$
Require $i \notin REG_X$
$(m_{in}, dec) \leftarrow\!\!\$\ \mathsf{GS.IssueX}(gmsk, m_{in} : st_{X,i})$
If $dec = \mathtt{accept}$ then
    $pk \leftarrow st_{X,i}$ ; $sk \leftarrow P_X[i]$
    Require $\mathsf{GS.wellformed}_X(\lambda, pk, sk)$
    $REG_X[i] \leftarrow (pk, sk)$
Return $(m_{in}, dec)$

$\underline{\textsc{Verify}(j, m, \sigma, L)}$
Require $j \in REG_{OA}$
$(oapk, oask) \leftarrow REG_{OA}[j]$
$b \leftarrow \mathsf{GS.Ver}(gmsk, oapk, L, m, \sigma)$
Return $b$
```

Figure 12: Traceability game for keyed-verification multi-opener group signatures.

Theorem 2. *Let Π_{GS} be the keyed-verification, multi-opener group signature scheme defined in Figure 8 over prime order p cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$. Let $\mathsf{MAC}_{\mathsf{GGM}}$ be the keyed-verification anonymous credentials scheme from [22] on $\mathbb{G}_1$. Then for any adversary $\mathcal{A}$ against the traceability of Π_{GS}, we give adversaries $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\text{trace}}_{\Pi_{\mathsf{GS}},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{unf}}_{\mathsf{MAC}_{\mathsf{GGM}},\mathcal{B}}(\lambda)$$

where $\mathcal{A}$ makes at most q_u and q_{ver} queries to the add user, and verify oracles, respectively, and $\mathcal{B}$ makes at most q_u and q_{ver} queries to its issue and show verify oracles, respectively.

Proof. We bound the advantage of adversary $\mathcal{A}$ by constructing an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to win the KVAC unforgeability game [22, Definition 6] whenever $\mathcal{A}$ wins the traceability game. Adversary $\mathcal{B}$ simulates the traceability game for $\mathcal{A}$. The issuer parameters from the KVAC unforgeability game are set as $gmpk$, and the $\textsc{Issue}$ and $\textsc{ShowVerify}$ oracles are used to simulate the actions of the group manager in $\textsc{AddU}$ and $\textsc{Verify}$.

To simulate issuing a signing key in $\textsc{AddU}$, $\mathcal{B}$ makes a call to the $\textsc{Issue}$ oracle to generate a MAC t and proof π of wellformedness (lines 23-30 of Figure 8). To make a call to $\textsc{Issue}$, $\mathcal{B}$ must send the secret signing key usk. This is fine since $\mathcal{B}$ only needs to properly simulate $\textsc{AddU}$ if a wellformed secret key has been added via $\textsc{KoskX}$, otherwise $\mathcal{B}$ will return $\perp$.

To simulate $\textsc{Verify}$, $\mathcal{B}$ runs its $\textsc{ShowVerify}$ oracle on σ with the following added MAC relation ϕ. The $\textsc{ShowVerify}$ oracle will calculate keyed-verifier values and run the verification procedure for $\mathcal{R}_2$ (lines 83-86 in Figure 8). The remainder of $\textsc{Verify}$, i.e. checking against the revocation list, can be run directly by $\mathcal{B}$.

$$\phi(y) = ct_1 = g_1^{\alpha_{ct}} \wedge ct_2 = g_1^y Z^{\alpha_{ct}} \wedge\ M_1 = g_1^{r_m} \wedge M_2 = g_2^{r_m} \wedge N_1 = g_1^{r_n} \wedge N_2 = g_2^{r_n} \wedge\ T_1 = M_1^{\alpha_T} \wedge T_2 = W^y N_1^{\alpha_T}\ .$$

If $\mathcal{A}$ wins the game, then $b_{ver} = 1$ meaning verification passed. This tells us two things. First, open did not return $\perp$, since the only way for open to return $\perp$ is if the signature proof verification fails; this cannot be the case since it is also checked by the verification algorithm. This means that open returned a $upk \notin REG_U$. Second, the signature σ verified under relation $\phi(y)$, where it was claimed that $ct_1 = g_1^{\alpha_{ct}} \wedge ct_2 = YZ^{\alpha_{ct}}$ for some $Y = g_1^y$. However, the call to the open algorithm returned a $upk = Y = ct_2/ct_1^z \notin REG_U$ for all y that credentials were issued for. The signature is then an example of a credential show for which the verification passes but $\phi(y) = 0$ allowing $\mathcal{B}$ to win the KVAC unforgeability game. □

D.4 Non-frameability

Non-frameability is defined by the game $\mathrm{G}^{\text{nf}}_{\mathsf{GS},\mathcal{A}}$ shown in Figure 13. We define the advantage of adversary $\mathcal{A}$ as:

$$\mathbf{Adv}^{\text{nf}}_{\mathsf{GS},\mathcal{A}}(\lambda) = \Pr\left[\mathrm{G}^{\text{nf}}_{\mathsf{GS},\mathcal{A}}(\lambda) = 1\right]$$

We say that a verifier-local revocable, keyed-verification, multi-opener group signature GS is *non-frameable* if $\mathbf{Adv}^{\text{nf}}_{\mathsf{GS},\mathcal{A}}(\cdot)$ is negligible for any polynomial-time adversary $\mathcal{A}$.

The non-frameability game is similar to the traceability game in that the adversary's goal is to output a signature with unwanted opening behavior. However, in the non-frameability game, we consider a stronger adversary that actively controls the platform,

```
$\underline{\text{Game } \mathrm{G}^{\mathrm{nf}}_{\mathsf{GS},\mathcal{A}}(\lambda)}$
$pp \leftarrow\!\!\$\ \mathsf{GS.Setup}(\lambda)$
$(gmpk, gmsk) \leftarrow\!\!\$\ \mathcal{A}(\perp : st_{\mathcal{A}})$
Require $\mathsf{GS.wellformed}_{\mathrm{GM}}(\lambda, gmpk, gmsk)$
$(j, m, \sigma) \leftarrow\!\!\$\ \mathcal{A}^{\text{WREGOA},\text{ADDX},\text{SIGN},\text{OPENREVOKE},\text{SKX}}(\perp : st_{\mathcal{A}})$
Assert $j \in H_{OA}$ ; $(oapk, oask) \leftarrow H_{OA}[j]$
$b_{ver} \leftarrow \mathsf{GS.Ver}(gmsk, oapk, RL[j], m, \sigma)$
$upk \leftarrow \mathsf{GS.Open}(oask, gmpk, m, \sigma)$
$b_{opn2} \leftarrow upk \in H_U \wedge upk \notin K_U$
Return $(upk, m) \notin \mathcal{Q} \wedge b_{ver} \wedge b_{opn2}$

$\underline{\text{WREGOA}(i, pk, sk)}$
Require $\mathsf{GS.wellformed}_{\mathrm{OA}}(\lambda, pk, sk)$
If $i \in H_{OA}$ then $(pk, sk) \leftarrow H_{OA}$
$REG_{OA}[i] \leftarrow (pk, sk)$

$\underline{\text{SIGN}(i, j, m)}$
Require $i \in H_U \wedge j \in REG_{OA}$
$(upk, usk) \leftarrow H_U[i]$ ; $(oapk, oask) \leftarrow REG_{OA}[j]$
$\sigma \leftarrow\!\!\$\ \mathsf{GS.Sign}(usk, gmpk, oapk, m)$
$\mathcal{Q} \mapsfrom (upk, m)$
Return $\sigma$

$\underline{\text{ADDX}(i, m_{in})_{X \in \{U, OA\}}}$
Require $i \notin H_X$
$(m_{in}, dec) \leftarrow\!\!\$\ \mathsf{GS.JoinX}(gmpk, m_{in} : st_X[i])$
If $dec = \texttt{accept}$ then
    $(pk, sk) \leftarrow st_X[i]$ ; $H_X[i] \leftarrow (pk, sk)$
Return $(m_{in}, dec)$

$\underline{\text{SKX}(i)}$
$K_X \mapsfrom i$
Return $H_X[i]$

$\underline{\text{OPENREVOKE}(m, \sigma, j)}$
Require $j \in H_{OA}$
$(oapk, oask) \leftarrow H_{OA}[j]$
$upk \leftarrow \mathsf{GS.Open}(oask, gmpk, m, \sigma)$
$\tau_R \leftarrow\!\!\$\ \mathsf{GS.Revoke}(oask, upk)$
Return $\tau_R$
```

Figure 13: Non-frameability game for keyed-verification multi-opener group signatures.

similar to the anonymity game. In the non-frameability game, the adversary wins if the signature opens to an honest user not controlled by the adversary, i.e., creates a successful forged signature. The adversary may register honest users and opening authorities using ADDX and may corrupt parties to learn their secret key through SKX. The adversary can generate signatures for uncorrupted users using SIGN and generate revocation tokens on arbitrary signatures using OPENREVOKE. After interacting with these oracles, the adversary outputs a message, signature pair and revocation list. The adversary wins if the message, signature pair was not previously output from SIGN, the signature verifies, and the open algorithm returns the public key of an uncorrupted user.

Discussion. Since the non-frameabilty game captures an adversary with similar power to that of the anonymity game, we make many of the same game design decisions. See the discussion in Section D.2 for more details.

Theorem 3. *Let Π_{GS} be the keyed-verification, multi-opener group signature scheme defined in Figure 8 over prime order p cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$. Let $\mathsf{MAC}_{\mathsf{GGM}}$ be the keyed-verification anonymous credentials scheme from [22] on $\mathbb{G}_1$. Then for any adversary $\mathcal{A}$ against the non-frameability of Π_{GS}, we give adversaries $\mathcal{A}_1$ to $\mathcal{A}_4$ such that*

$$\mathbf{Adv}^{\mathrm{nf}}_{\Pi_{\mathsf{GS}},\mathcal{A}}(\lambda) \leq q_u\big(\mathbf{Adv}^{\mathrm{sound}}_{\mathsf{NiZK}_{\mathcal{R}_1},\mathcal{A}_1,\mathcal{X}_{\mathcal{R}_1}}(\lambda) + \mathbf{Adv}^{\mathrm{anon}}_{\mathsf{MAC}_{\mathsf{GGM}},\mathcal{A}_2,\mathcal{S}_{\mathsf{MAC}}}(\lambda) + \mathbf{Adv}^{\mathrm{simext}}_{\mathsf{SoK}_{\mathcal{R}_2},\mathcal{A}_3,\mathcal{X}_{\mathcal{R}_2},\mathcal{S}_{\mathcal{R}_2}}(\lambda) + \mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G}_1,p,\mathcal{A}_4}(\lambda)\big)$$

where $\mathcal{A}$ makes at most q_u queries to the add user oracle.

Proof. We bound the advantage of $\mathcal{A}$ by bounding the advantage of each of a series of game hops. Similarly to as in the anonymity proof, we define $\mathrm{G} = \mathrm{G}^{\mathrm{nf}}_{\Pi_{\mathsf{GS}},\mathcal{A}}(\lambda)$ and define games $\mathrm{G_A}$, $\mathrm{G_B}$, $\mathrm{G_C}$, and $\mathrm{G_D}$ that slowly transform the view of the adversary so that signing queries for a guessed user are no longer dependent on their secret key. Then we will show in the final game $\mathrm{G_D}$, if $\mathcal{A}$ wins, we can win the discrete logarithm game. The inequality above follows from simple calculations based on the following claims which we will justify:

(1) $\mathbf{Adv}^{\mathrm{nf}}_{\Pi_{\mathsf{GS}},\mathcal{A}}(\lambda) = \Pr[\mathrm{G} = 1] \leq q_u \cdot \Pr[\mathrm{G_A} = 1]$

(2) $|\Pr[\mathrm{G_A} = 1] - \Pr[\mathrm{G_B} = 1]| = \mathbf{Adv}^{\mathrm{sound}}_{\mathsf{NiZK}_{\mathcal{R}_1},\mathcal{A}_1,\mathcal{X}_{\mathcal{R}_1}}(\lambda)$

(3) $|\Pr[\mathrm{G_B} = 1] - \Pr[\mathrm{G_C} = 1]| = \mathbf{Adv}^{\mathrm{anon}}_{\mathsf{MAC}_{\mathsf{GGM}},\mathcal{A}_2,\mathcal{S}_{\mathsf{MAC}}}(\lambda)$

(4) $|\Pr[\mathrm{G_C} = 1] - \Pr[\mathrm{G_D} = 1]| = \mathbf{Adv}^{\mathrm{simext}}_{\mathsf{SoK}_{\mathcal{R}_2},\mathcal{A}_3,\mathcal{X}_{\mathcal{R}_2},\mathcal{S}_{\mathcal{R}_2}}(\lambda)$

(5) $\Pr[\mathrm{G_D} = 1] = \mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G}_1,p,\mathcal{A}_4}(\lambda)$

Claim 1: Without loss of generality, assume calls to $\textsc{AddX}$ are made with incrementing indices, e.g., $i = 1, 2, \ldots, q$. $\mathrm{G_A}$ is the same as G except it guesses the signing party i on which $\mathcal{A}$'s winning signature will open to and aborts if it is incorrect. If $\mathcal{A}$ does not win, or if it wins by opening to a upk that does not belong to user i, then $\mathrm{G_A}$ sets a bad_A flag and aborts. This also means $\mathrm{G_A}$ aborts if party i is queried to SKU since $\mathcal{A}$ cannot win on a corrupted user. By an identical-until-bad argument and the fundamental lemma of game playing [8], we have that

$$\Pr[\mathrm{G} = 1 \wedge \neg \mathsf{bad}_\mathrm{A}] = \Pr[\mathrm{G_A} = 1 \wedge \neg \mathsf{bad}_\mathrm{A}] \,.$$

And since $\mathrm{G_A}$ aborts and outputs 0 when bad_A is set, we have

$$\Pr[\mathrm{G} = 1 \wedge \neg \mathsf{bad}_\mathrm{A}] = \Pr[\mathrm{G_A} = 1] \,.$$

Then, we have

$$\begin{aligned} \Pr[\mathrm{G_A} = 1] &= \Pr[\mathrm{G} = 1 \wedge \neg \mathsf{bad}_\mathrm{A}] \\ &= \Pr[\neg \mathsf{bad}_\mathrm{A}] \cdot \Pr[\mathrm{G} = 1] \end{aligned} \tag{1}$$

where (1) holds because the condition to set bad_A is independent of the rest of the game G.

Lastly since the party is guessed at random, the probability that the guess is correct and bad_A is not set is at least

$$\Pr[\neg \mathsf{bad}_\mathrm{A}] \geq \frac{1}{q_u} \,.$$

Sketch: The arguments and game hops for claims 2-4 follow analogously to the same claims in the anonymity proof. We refer the reader to the details there (Section D.2).

Claim 5: Observe that in $\mathrm{G_D}$, the secret key y of signing user i is not used. Yet to win $\mathrm{G_D}$, the adversary $\mathcal{A}$ must produce a verifying signature that opens to Y. Since the extractor for the signature proof did not fail, we have that it will correctly extract y where $Y = g^y$. We build an adversary $\mathcal{A}_4$ for the discrete logarithm game that wins whenever $\mathcal{A}$ wins by setting the signing user's public key Y to the discrete logarithm challenge element and returning the extracted value y.

□

E Providing Message Binding for One-time Tokens

In this section, we describe an alternate token showing protocol to provide ciphertext binding for senders, preventing the DoS attack described in Section 5. At a high level, our alternate proposal can be thought of as receiving a MAC on a one-time-use BLS signature public key. Messages are bound to a token by signing the message under the BLS signature scheme [14].

Recall that senders mint tokens of the form (ν, t) where $(u_0, u_1 = u_0^{x_0 + x_1 \nu}) \leftarrow t$ is a valid MAC for m under the $\mathsf{MAC_{GGM}}$ scheme [22]. Our alternate token showing protocol will make use of the fact that the platform can verify the MAC t on g_2^ν instead of ν. Here we use g_2^ν as a one-time BLS verification key and ν as the secret signing key known only to the sender. The sender will sign their ciphertext with the BLS signing key, meaning that the platform will not be able to swap out the ciphertext. It will make use of a hash function $\mathsf{H} : * \rightarrow \mathbb{G}_2$.

If a sender wants to send a ciphertext ct with token $(\nu, t = (u_0, u_1))$, they will "show" the token by constructing and sending $(K, H) = (g_2^\nu, \mathsf{H}(ct)^\nu)$ along with (ct, t). The BLS signature H is what binds the ciphertext as it can only be created from knowledge of ν.

The platform will verify the token by first checking the validity of the MAC tag t against BLS verification key K,

$$e(u_0, g_2^{x_0} K^{x_1}) \stackrel{?}{=} e(u_1, g_2) \,,$$

and then checking the binding of the ciphertext with the BLS signature,

$$e(\mathsf{H}(ct), K) \stackrel{?}{=} e(H, g_2) \,.$$

If both these checks pass, the platform accepts the token, adding K to the strikelist and forwarding (K, H) and ct to the recipient. Even though ν is not shown to the platform, a token can still only be spent once since the one-time verification key $K = g_2^\nu$ uniquely maps ν. Note, the recipient can perform the signature verification to ensure the ciphertext was created by the party that knows the signing key of K.

The recipient needs to be able to identify the sender based on $K = g_2^\nu$. To do this, we change the ciphertext created for the recipient during the minting process to encrypt g_2^ν instead of g_1^ν. This requires the recipient to publish a public key in $\mathbb{G}_2$, but otherwise the minting protocol does not change.

We leave to future work the formal analysis of this extension. In some sense, this proposal can be thought of as extending the algebraic MAC protocol of Chase et al. [22] to support group elements in the message space (as opposed to only scalars) and support verification using a bilinear pairing. An alternative tack to avoid pairings would be to use the algebraic MAC protocol of Signal [23], which supports group elements in the message space without pairings (and a pairing-free signature scheme, like Schnorr).