Avoiding Lock Outs: Proactive FIDO Account Recovery using Managerless Group Signatures

Sunpreet S. Arora, Saikrishna Badrinarayanan, Srinivasan Raghuraman, Maliheh Shirvanian, Kim Wagner and Gaven Watson

Visa Research

Abstract

Passwords are difficult to remember, easy to guess and prone to hacking. While there have been several attempts to solve the aforementioned problems commonly associated with passwords, one of the most successful ones to date has been by the Fast Identity Online (FIDO) alliance. FIDO introduced a series of protocols that combine local authentication on a user device with remote validation on relying party servers using public-key cryptography.

One of the fundamental problems of FIDO protocols is complete reliance on a single user device for authentication. More specifically, the private key used for signing relying party challenges can only be stored on a single device. Each FIDO authenticator key is linked uniquely to an account with a relying party service. As a result a lost or stolen user device necessitates creation of new user account, using a new device, with each (previously enrolled) relying party service.

To overcome this limitation, we introduce a dynamic managerless group signature scheme that organizes authenticators into groups. Each authenticator in a group has a unique private key that links it to an account with a relying party, which can sign relying party challenges. The relying party server has a group verification key that can validate challenges signed using the private key of any authenticator in a group. Our approach provides additional redundancy and usability to the FIDO protocol whilst still achieving the security properties expected in the FIDO setting such as unforgeability and unlinkability.

1 Introduction

"Nobody likes passwords. They are inconvenient, insecure, and expensive."

The aforementioned quote from Microsoft's Security Team [42] succinctly summarizes the problem of passwords. Given the challenges associated with

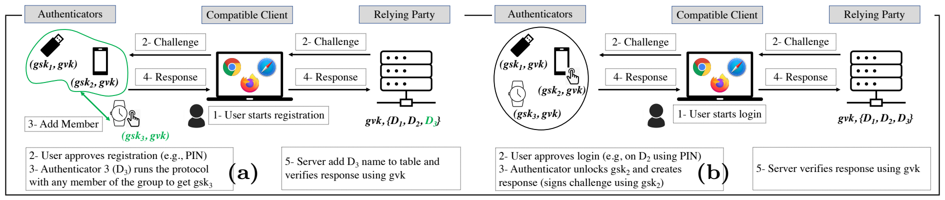

Authenticators
Compatible Client
Relying Party
2- Challenge
4- Response
2- Challenge
4- Response
(gsk_1, gvk)
(gsk_2, gvk)
(gsk_3, gvk)
3- Add Member
1- User starts registration
gvk, {D_1, D_2, D_3}
2- User approves registration (e.g., PIN)
3- Authenticator 3 (D_3) runs the protocol with any member of the group to get gsk_3
(a)
5- Server add D_3 name to table and verifies response using gvk
Authenticators
Compatible Client
Relying Party
2- Challenge
4- Response
2- Challenge
4- Response
(gsk_1, gvk)
(gsk_2, gvk)
(gsk_3, gvk)
1- User starts login
gvk, {D_1, D_2, D_3}
2- User approves login (e.g, on D_2 using PIN)
3- Authenticator unlocks gsk_2 and creates response (signs challenge using gsk_2)
(b)
5- Server verifies response using gvk

Figure 1: Proposed managerless group signature scheme for proactive FIDO account recovery. Fig (a) Registration: each authenticator can add a new authenticator to the group. Fig (b) Authentication: each authenticator has its own signing key gsk that is used to sign a challenge c at the time of authentication. The FIDO server S has a single group verification key gvk that is used to verify c signed by any gsk.

passwords, there has been a big push in the authentication industry toward *passwordless authentication* [42]. Eliminating passwords altogether is a challenging problem given their legacy in authentication systems and the associated consumer habituation. Biometrics-based authentication solutions, e.g., TouchID [4] and FaceID [3], address the conundrum of security and convenience in the context of local authentication to applications on a consumer device but still rely on passwords as a backup. Another key trend in the authentication industry is that of two-factor (2FA) and multi-factor authentication (MFA) [36]. Hence, relying parties are increasingly leveraging 2FA/MFA solutions offered by service providers (e.g., Google [28], Microsoft [34] and HID [29]) for securing financial transactions and consumer accounts.

One of the most successful initiatives in the authentication industry is being undertaken by the Fast Identity Online (FIDO) Alliance [22]; an industry consortium focused on developing open authentication standards with the goal of reducing the world's reliance on passwords. FIDO introduced a series of standards and specifications targeted toward passwordless [21] and 2FA/MFA [20] that combine native authentication on a user device (e.g., using TouchID, FaceID) with public-key cryptography techniques for server-side validation. A fundamental problem with FIDO protocols is complete reliance on the storage of a single copy of the private key used for signing relying party challenges. Account recovery in case the device is lost, stolen or inaccessible is, therefore, a critical challenge to address for widespread FIDO adoption.

1.1 The Problem of Account Recovery

Account recovery is a crucial component of any web-based service. Most recovery processes involve retrieval or reset of an account credential such as a password or secret key used to secure the account. Password managers, e.g., [16, 1, 31] are one of the most popular methods to store and manage such credentials. Here, a cloud-based password vault is generally protected using a *master* password. In case a user loses their account credentials, the password vault can be restored from the cloud using the master password.

Compared to password managers, a device or cloud-based hardware approach provides a more secure alternative. An example of a device-based approach is the Secure Enclave in Apple devices which enables a secure vault called KeyChain [5] to securely store website passwords, credit card, and other secret information. A backup of the Keychain can be stored in the cloud such that the information is end-to-end encrypted using a secret key that is unique to the device and derived using the device passcode. A recent approach called Safetypin [17] provides additional fault tolerance by distributing encrypted credentials backups across a collection of cloud-based hardware security modules. A PIN can be used to recover the credentials on the device.

1.2 Account recovery in FIDO

Account recovery in the context of FIDO authentication is fundamentally different from other forms of authentication such as passwords and 2FA. While passwords and other secret keys can be stored in a cloud-based password vault and/or device or cloud-based secure hardware devices, FIDO secret keys are device dependant and cannot be backed-up directly or encrypted to a secondary device or the cloud. In the 2019 whitepaper titled "Recommended Account Recovery Practices for FIDO Relying Parties" [27], the FIDO alliance recommends two strategies to overcome this limitation: (i) use of identity proofing or user onboarding methods (e.g., in-person or remote KYC) for account recovery. (ii) use of multiple client authenticators per account to reduce the likelihood of account recovery. The latter is discussed further in a follow-up whitepaper by FIDO [37]. More recently, FIDO introduced the concept of multi-device credentials which allows FIDO secret keys to be synced across devices using a platform provider's cloud [2]. Apple recently introduced a solution based on this concept called Passkeys [6] which lets a user sync their FIDO credentials using iCloud. Table 3 given in Appendix B compares prevailing FIDO account recovery approaches.

Our goal is to enable multiple authenticators per user account for account recovery. However, unlike aforementioned solutions (i) the relying party only needs to hold one public key per account which reduces the verification costs compared to earlier solutions while still satisfying the unlinkability properties expected in FIDO, (ii) the addition and removal of authenticators can be managed by a user on a client device without a strict dependency on a relying party, (iii) and an authenticator group can be used with multiple relying parties thereby reducing the burden on the user to register additional authenticators separately with each relying party.

1.3 Our Contributions

We address the problem of account recovery via a protocol that permits a user to enroll a group of authenticators. Any one of these authenticators can then be used for authentication (see Figure 1). The protocol permits a user to add or remove authenticators from an authenticator group at any time.

A naïve solution for account recovery would extract an authenticator's secret signing key and share the key across all members of the group. This key sharing, however, has significant security issues. Our approach instead utilizes a special type of signature scheme known as a group signature [7, 9, 11, 12]. Such a scheme permits each member of a group to individually sign messages using their own *unique* secret key. The signed message can then be verified as being signed by the group using a public key associated with that group. We provide further background on group signature schemes in Appendix C.

State-of-the-art group signature schemes rely on a privileged member that is responsible for maintaining the structure of the group called the Manager. The manager holds the secret information necessary for adding or removing group members. As a result, there is complete dependency on the manager for recovery. If the manager device is lost or stolen, so too is the ability to recover/change the group.

Democratic Group Signatures [33] were introduced to address the setting of a managerless group signature scheme. However, they require all members to participate in order to add a new member, thus making addition of new members as expensive as a setup. In addition, the addition of a new member changes the group verification key.

1) Introducing Managerless Group Signatures in Section 2, 3: We introduce the first Managerless Group Signature (MGS) scheme which avoids challenges related to using standard group signature schemes when applied to FIDO account recovery. In an MGS scheme, each member of a group has the ability (through collaboration) to change the group structure. We formalize the proposed scheme by providing security definitions for properties relevant to group signatures, such as Unforgeability, Anonymity, Traceability and Revocability, in the managerless setting, some of which (such as traceability) are of interest to applications beyond FIDO authentication. In our setting, unlike democratic group signatures, each member can individually or collaboratively add a new member, and addition of the new members does not change the verification key.

2) FIDO Account Recovery using MGS in 4 and Section 5: We can apply MGS within the FIDO context and achieve the required security properties of unforgeability and unlinkability.

Our protocol design has minimal impact on the existing FIDO protocol. The message flow during authentication as well as the user experience during FIDO authentication remain the same. The only fundamental change to the core FIDO authentication protocol is the underlying signature scheme that is used to support device groups. The signing and verification algorithms that are introduced in the proposed solution are, however, direct plug-in replacements for the core algorithms used in commercial FIDO deployments. We discuss how this MGS-based FIDO achieves the unforgeability and unlinkability properties specified by The FIDO Security Reference document [19].

3) Implementation and Performance Evaluation in Section 6: We implement and provide a performance evaluation of this enhanced MGS-based FIDO protocol. Our evaluation shows that our cryptographic operations are light-weight (about 20-100ms depending on the signature scheme and curve type) and only slightly slower than ECDSA used in FIDO2. More importantly, adding a new authenticator has a negligible delay ($\approx$0ms) and does not increase verification cost. This is in contrast to FIDO, where adding a new authenticator requires running key generation in collaboration with a relying party, and the verification cost increases linearly as the number of authenticators increases.

4) MGS with Full Security in Section 7: Finally we present an MGS constructions which achieves all the properties of group signature scheme, including traceability. While such properties are not needed in the FIDO setting, they are not only of theoretical interest but also desirible additional properties to achieve.

2 Signing Key Re-randomizable Signatures

Let us first revisit traditional signature schemes and discuss how they can be extended with additional properties that will be useful to define the proposed scheme. A hash function is typically used to process messages prior to the messages being signed. Let $\texttt{Hash} : \{0,1\}^* \to \mathbb{Z}_q$ be such a collision-resistant hash function.

In what follows we focus on the Schnorr signature scheme [39] but similar extensions also hold for the El Gamal signature scheme [24]. Further details on this are provided in Appendix E.

Let $\sum_{\mathsf{Schnorr}}$ be the **Schnorr signature scheme** [39] over a group $\mathbb{G}$ of prime order $q = \mathcal{O}(2^\lambda)$ with generator g, consisting of the following algorithms:

- $\texttt{KeyGen}(1^\lambda)$: Samples a random $x \leftarrow \mathbb{Z}_q$ and sets $\mathsf{sk} = x$ and $\mathsf{vk} = g^x$.
- $\texttt{Sign}(m; \mathsf{sk} = x)$: Samples a random $r \leftarrow \mathbb{Z}_q$ and sets $e = \texttt{Hash}(g^r \| m)$, $s = r - xe$ and $\sigma = (s, e)$.
- $\texttt{Ver}(m, (s, e); \mathsf{vk})$: Checks if $e = \texttt{Hash}(g^s \mathsf{vk}^e \| m)$.

$\sum_{\mathsf{Schnorr}}$ is secure in the random oracle model based on the discrete logarithm assumption [40].

2.1 Re-randomizable Signature Scheme

Consider the notion of expanding the space of signing keys in order to have an arbitrarily large number of functionally equivalent signing keys for any given verification key. In this context, it would be desirable to generate a random functionally equivalent signing key given any one signing key corresponding to a verification key. To do so, we describe a generic ideal functionality $\mathcal{F}_{\mathbf{ReRand}}$ in Figure 2.

We show that by extending the signing key space of the signature scheme in a simple way, we can indeed achieve these properties. Let $M = 2^{\mathcal{O}(\lambda)}$ be a scaling parameter (that is exponentially large in λ). The overall idea is that instead of working modulo q, we will work modulo Mq. We elaborate on this in more detail below. Also, note that the techniques described in the following

$\mathcal{F}_{\mathbf{ReRand}}$

A has $(\mathsf{vk}, \mathsf{sk}_\mathsf{A}) \leftarrow \sum.\mathtt{KeyGen}(1^\lambda)$ for a signature scheme $\sum$.
B wants a random signing key sk_B corresponding to vk.

Inputs: $\mathsf{pub} = \mathsf{vk}$, $\mathsf{inp}_\mathsf{A} = \mathsf{sk}_\mathsf{A}$, $\mathsf{inp}_\mathsf{B} = \perp$

Outputs: $\mathsf{outp}_\mathsf{B} = \mathsf{sk}_\mathsf{B}$ uniformly distributed subject to the constraint that, for all m, r:
$\sum.\mathtt{Ver}\ (m, \sum.\mathtt{Sign}(m, \mathsf{sk}_\mathsf{B}; r), \mathsf{vk}) = 1$.

Figure 2: Ideal Functionality $\mathcal{F}_{\mathbf{ReRand}}$

section can be used to enhance most group-based signature schemes with the property of signing-key re-randomization.

2.2 Re-randomizable Schnorr Signatures

Let $\sum^*_{\mathsf{Schnorr}}$ be the **modified Schnorr signature scheme** over a group $\mathbb{G}$ of prime order $q = \mathcal{O}(2^\lambda)$ with generator g, consisting of the following algorithms:

- $\mathtt{KeyGen}(1^\lambda)$: Samples a random $x \leftarrow \mathbb{Z}_{Mq}$ and sets $\mathsf{sk} = x$ and $\mathsf{vk} = g^{x \mod q}$.
- $\mathtt{Sign}(m, \mathsf{sk} = x)$: Samples a random $r \leftarrow \mathbb{Z}_q$ and sets $e = \mathtt{Hash}(g^r \| m)$, $s = (r - xe) \mod q$ and $\sigma = (s, e)$.
- $\mathtt{Ver}(m, (s, e), \mathsf{vk})$: Checks if $e = \mathtt{Hash}(g^s \mathsf{vk}^e \| m)$.

Figure 3 describes a protocol Π_{Schnorr} that securely realizes $\mathcal{F}_{\mathbf{ReRand}}$ for the above scheme. The security of Π_{Schnorr} is easy to observe from the correctness of the modified Schnorr signature scheme. We also prove that the modified signature scheme is existentially unforgeable in Appendix E.

3 Managerless Group Signatures (MGS)

We define a dynamic managerless group signature scheme where members of the group can collectively function (as enforced by a group structure) as managers – for issuing and/or revoking – in addition to being able to generate signatures on data. A formal definition of the syntax of such a scheme is provided in Section 3.1. For security guarantees, we draw from the various security properties satisfied by traditional manager-based group signature schemes (see Appendix C), e.g., to achieve some form of full-anonymity and full-traceability as allowed for by the group structure.

Finally, security against dishonest managers would translate to some form of security against collusions of subsets of parties as dictated by the group structure. Similar to prior works [9, 10], we opt for a game-based definition of security. See Section 3.2 for detailed description of the security games.

Π_{Schnorr}

A has a key-pair $(\mathsf{vk}, \mathsf{sk_A}) \leftarrow \sum^*_{\mathsf{Schnorr}}.\mathtt{KeyGen}(1^\lambda)$
B wants a random signing key $\mathsf{sk_B}$ corresponding to vk.

Inputs: $\mathsf{pub} = (\mathbb{G}, q, M, \mathsf{vk})$, $\mathsf{inp_A} = \mathsf{sk_A}$, $\mathsf{inp_B} = \perp$

Protocol:

1. A computes and sends $x = (\mathsf{sk_A} \mod q)$ to B.
2. B samples random $y \leftarrow \mathbb{Z}_M$.
3. B outputs $\mathsf{sk_B} = x + yq$.

Outputs: $\mathsf{outp_A} = \perp$, $\mathsf{outp_B} = \mathsf{sk_B}$

Figure 3: Protocol Π_{Schnorr} for realizing $\mathcal{F}_{\mathbf{ReRand}}$ in the modified Schnorr signature scheme.

3.1 Syntax Definition

We define a managerless group signature scheme $\sum^{\mathsf{w/o\ manager}}_{\mathsf{Group}}$ for group G with respect to a group structure Π. Since each of the operations for a managerless group signature scheme may potentially involve interaction among multiple members of the group, we define them as protocols as opposed to algorithms. Formally, it consists of the following *protocols*:

- $\mathtt{KeyGen}(1^\lambda, G, \Pi)$: Samples a public group verification key gvk, a public identity ledger Table and private keys for each of the individual group members $\{\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i\}_{i \in G}$ where gsk denotes a group signing key, ik denotes an issuing key, ok denotes an opening key and rk denotes a revocation key.

- $\mathtt{AddMember}(j; \{\mathsf{ik}_i\}_{i \in X})$: Adds a new member j to G (updating Table,[1] G and Π accordingly) and generates a private key $\{\mathsf{gsk}_j, \mathsf{ik}_j, \mathsf{ok}_j, \mathsf{rk}_j\}$ for the new member of the group, assuming the members of set X satisfy Π.

- $\mathtt{Sign}(m; \{\mathsf{gsk}_i\}_{i \in X})$: Generates a signature σ assuming the members of set X satisfy Π.

- $\mathtt{SigOpen}(m, \sigma; \{\mathsf{ok}_i\}_{i \in X})$[2]: Determines the members Y of the group who generated σ, assuming the members of set X satisfy Π.

- $\mathtt{RevokeMember}(j; \{\mathsf{rk}_i\}_{i \in X})$: Revokes an existing member $j \in G$ from G (up-

[1] $\mathtt{AddMember}$ will add an entry to Table without affecting any of the other entries in the ledger. Furthermore, the new entry also identifies the set X that was involved in adding the new member j, whose identity is however hidden.

[2] This algorithm may be augmented to also produce a "proof" π that can be (publicly) verified by a "judging" algorithm $\mathtt{Judge}$.

dating Table,[3] G and Π accordingly), assuming the members of set X satisfy Π.

- $\mathtt{Ver}(m, \sigma; \mathsf{gvk})$: Outputs 0 or 1.

We define correctness of such a scheme informally: for all honestly generated parameters, honestly generated signatures always verify, and can be opened using the allowed sets of opening keys. The security guarantees provided by the scheme are also described.

3.2 Security Definition

Intuitively, we would like to capture the properties of unforgeability anonymity, traceability and revocability. We opt for a game-based security definition, where we define a security game played between a challenger and an adversary, who will attempt to break one of the properties listed above. We first describe the guarantees associated with each of these properties.

Unforgeability requires that an adversary is not able to produce a valid message-signature pair that it cannot trivially create (for instance, using the keys of the corrupt parties or replaying a pair generated by honest parties).

Anonymity requires that an adversary is not able to distinguish between signatures generated by different sets of parties involving honest parties, except trivially, which could be by possessing enough keys to open signatures, or revoking parties so as to disambiguate signatures by seeing how `Ver`'s behavior differs pre- and post-revocation.

Revocability requires that an adversary is not able to generate a valid signature that opens to a set containing a revoked member.

Traceability requires that an adversary is not able to generate a valid signature that cannot be traced back correctly to the signers who created it even if it is so permitted by the group structure Π.

With the above intuitive descriptions of the security properties, we proceed to define the security games formally below. Note that we define separate security games for each of the properties described above. Only the definitions for unforgeablity and revocability will be needed to achieve security in the FIDO setting, while the notions of anonymity and traceability are of further theoretical interest in the context of group signatures.

Syntax of the games. Each security game for the group signature scheme $\sum = \sum_{\mathsf{Group}}^{\mathsf{w/o\ manager}}$ will be played between a challenger $\mathcal{C}$ and a probabilistic polynomial time (PPT) adversary $\mathcal{A}$. During the game, $\mathcal{A}$ interacts with $\mathcal{C}$ in a specified way (defined in Figure 6, Figure 7, Figure 8 and Figure 9), and based on this interaction, $\mathcal{C}$, at the end of the game, outputs either `True` (indicating that $\mathcal{A}$ won the game, or equivalently, that $\mathcal{A}$ broke the specific property of

[3] `RevokeMember` will flag an entry to Table as "revoked" without affecting any of the other entries in the ledger. Furthermore, the flag also identifies the set X that was involved in revoking the member j, whose identity is however hidden.

```
Oracle Add2U($i$)
if $i \in U$,   return $\perp$
else $U = U \cup \{i\}$,   $H = H \cup \{i\}$

Oracle Corrupt($i$)
if $i \notin H$,   return $\perp$
else $H = H \backslash \{i\}$,   return $(\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i)$

Oracle Add2G($j, X$)
if $j \notin U \backslash (G \cup R)$,   return $\perp$
elseif $X \nsubseteq G$ or $X \cap H = \emptyset$ or $(X, \mathsf{add}) \notin \Pi$,   return $\perp$
else
  $(\mathsf{gsk}_j, \mathsf{ik}_j, \mathsf{ok}_j, \mathsf{rk}_j) \leftarrow \sum.\texttt{AddMember}(j; \{\mathsf{ik}_i\}_{i \in X})$
  if $j \notin H$,   return $(\mathsf{gsk}_j, \mathsf{ik}_j, \mathsf{ok}_j, \mathsf{rk}_j)$

Oracle Append2G($j, X, (\mathsf{gsk}, \mathsf{ik}, \mathsf{ok}, \mathsf{rk})$)
if $j \notin U \backslash (G \cup R)$,   return $\perp$
elseif $X \nsubseteq G$ or $X \cap H \neq \emptyset$ or $(X, \mathsf{add}) \notin \Pi$,   return $\perp$
else
  $G = G \cup \{j\}$
  if $j \in H$,   $(\mathsf{gsk}_j, \mathsf{ik}_j, \mathsf{ok}_j, \mathsf{rk}_j) \leftarrow \sum.\texttt{AddMember}(j; \{\mathsf{ik}_i\}_{i \in X})$
  else $(\mathsf{gsk}_j, \mathsf{ik}_j, \mathsf{ok}_j, \mathsf{rk}_j) \leftarrow (\mathsf{gsk}, \mathsf{ik}, \mathsf{ok}, \mathsf{rk})$
```

Figure 4: Oracles that an adversary has access to in the security games (Part 1/2).

the scheme) or `False` (indicating that $\mathcal{A}$ lost the game). During the course of the game, $\mathcal{A}$ is allowed access to some specific set of oracles (among the ones described in Figure 4 and Figure 5). If $\mathcal{A}$ has access to an oracle, it may issue queries to it and obtain responses.

Notation. The universe of parties is U, the set of honest parties is $\mathcal{H}$ and the set of revoked parties is R. At the beginning of each of the games, these sets are initialized to be empty. During the course of the game, they may be updated (by the oracles). G will denote the parties who are (currently) part of the group and Π will denote the group structure. G and Π will be picked by $\mathcal{A}$ at the beginning of each game. Subsequently, G and Π may be updated (by the oracles).

Two other pieces of notation are List, which denotes the list of messages signed by the Sign oracle, and Chall, which denotes the challenge signature (for the anonymity game $\mathsf{Exp}^{\mathsf{anonymity}}_{\sum,\mathcal{A},\lambda}$ defined in Figure 7). More about List and Chall

```
Oracle Sign$(m, X)$
if $X \not\subseteq G$ or $X \cap H = \emptyset$ or $(X, \mathsf{sign}) \notin \Pi$,   return $\perp$
else
    $\sigma \leftarrow \sum.\mathtt{Sign}(m; \{\mathsf{gsk}_i\}_{i \in X})$
    $\mathsf{List} = \mathsf{List} \cup \{(m, \sigma, X)\}$
    return $\sigma$

Oracle SigOpen$(m, \sigma, X)$
if $X \not\subseteq G \cup R$ or $(X, \mathsf{open}) \notin \Pi$,   return $\perp$
elseif $\sum.\mathtt{Ver}(m, \sigma; \mathsf{gvk}) = 0$,   return $\perp$
else
    $Y \leftarrow \sum.\mathtt{SigOpen}(m, \sigma; \{\mathsf{ok}_i\}_{i \in X})$
    return $Y$

Oracle Remove2G$(j, X)$
if $X \not\subseteq G$ or $X \cap H = \emptyset$ or $(X, \mathsf{revoke}) \notin \Pi$,   return $\perp$
else
    $\sum.\mathtt{RevokeMember}(j; \{\mathsf{rk}_i\}_{i \in X})$
    $R = R \cup \{j\}$

Oracle Delete2G$(j, X)$
if $X \not\subseteq G$ or $X \cap H \neq \emptyset$ or $(X, \mathsf{add}) \notin \Pi$,   return $\perp$
else $G = G \backslash \{j\}$ and $R = R \cup \{j\}$
```

Figure 5: Oracles that an adversary has access to in the security games (Part 2/2).

can be found in the expositions ahead.
Oracles. Each game gives the adversary access to the following oracles:

- Add2U(i): To add the party with identifier i to set U. At this point, i is also added to H. New parties are honest by default, but can later be corrupted.

- Corrupt(i): To corrupt an existing honest (in H) party with identifier i. i gets removed from H. If there are any keys associated with i (such as $\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i$), they are handed over to $\mathcal{A}$.

- Add2G(j, X): To add an existing party with identifier j to the group G using the keys of parties in set X. The oracle requires that $X \cap H \neq \emptyset$ because, otherwise, the adversary can add the member using the keys it already has. The oracle runs $\sum.\mathtt{AddMember}(j; \{\mathsf{ik}_i\}_{i \in X})$ to obtain $\{\mathsf{gsk}_j, \mathsf{ik}_j, \mathsf{ok}_j, \mathsf{rk}_j\}$. If

```
$\mathsf{Exp}^{\mathsf{forgery}}_{\sum,\mathcal{A},\lambda}$
$U = \emptyset, H = \emptyset, R = \emptyset, \mathsf{List} = \emptyset$
$(G, \Pi, \mathsf{state}) \leftarrow \mathcal{A}^{\mathsf{Add2U},\mathsf{Corrupt}}(1^\lambda)$
**if** $G \not\subseteq U$,   **return** False
$(\mathsf{gvk}, \{\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i\}_{i \in G}) \leftarrow \sum.\mathtt{KeyGen}(1^\lambda, G, \Pi)$
$\mathsf{state} = (\mathsf{gvk}, \{\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i\}_{i \in G \setminus H})$
$(m^*, \sigma^*, X^*) \leftarrow \mathcal{A}^{Oracles}(\mathsf{state})$
$Z \leftarrow \sum.\mathtt{SigOpen}(m^*, \sigma^*; \{\mathsf{ok}_i\}_{i \in X^*})$
**if** $\sum.\mathtt{Ver}(m^*, \sigma^*; \mathsf{gvk}) = 0$  **return** False
**elseif** $(m^*, \sigma^*, \cdot) \notin \mathsf{List}$ **and** $Z \cap H \neq \emptyset$  **return** True
**else**  **return** False
```

Figure 6: Security game for unforgeability. *Oracles* denotes the set of oracles the adversary has access to, specifically $\{\mathsf{Add2U}, \mathsf{Corrupt}, \mathsf{Add2G}, \mathsf{Append2G}, \mathsf{Sign}, \mathsf{SigOpen}, \mathsf{Remove2G}, \mathsf{Delete2G}\}$.

$j \notin H$, it hands over $\{\mathsf{gsk}_j, \mathsf{ik}_j, \mathsf{ok}_j, \mathsf{rk}_j\}$ to $\mathcal{A}$.

- $\mathsf{Append2G}(j, X, \{\mathsf{gsk}, \mathsf{ik}, \mathsf{ok}, \mathsf{rk}\})$: To add the existing party with identifier j to the group G using the keys of parties in set X.[4] The oracle adds j to G. If $j \in H$, it additionally runs $\sum.\mathtt{AddMember}(j; \{\mathsf{ik}_i\}_{i \in X})$ to obtain $\{\mathsf{gsk}_j, \mathsf{ik}_j, \mathsf{ok}_j, \mathsf{rk}_j\}$. Otherwise, $\{\mathsf{gsk}, \mathsf{ik}, \mathsf{ok}, \mathsf{rk}\}$ are the keys for j.[5]

- $\mathsf{Sign}(m, X)$: To sign a message m using the keys of parties in set X. The oracle requires that $X \cap H \neq \emptyset$ because, otherwise, the adversary can sign the message using the keys it already has. The oracle runs $\sum.\mathtt{Sign}(m; \{\mathsf{gsk}_i\}_{i \in X})$ to obtain σ. The oracle adds σ (along with m and X) to List and hands over σ to $\mathcal{A}$.

- $\mathsf{SigOpen}(m, \sigma, X)$: To open a (valid) signature σ on a message m using the keys of parties in set X. The oracle runs $\sum.\mathtt{SigOpen}(m, \sigma; \{\mathsf{ok}_i\}_{i \in X})$ to obtain Y and hands over Y to $\mathcal{A}$. An important point to note is that in the anonymity game ($\mathsf{Exp}^{\mathsf{anonymity}}_{\sum,\mathcal{A},\lambda}$ defined in Figure 7), this oracle will not open the challenge signature Chall.

- $\mathsf{Remove2G}(j, X)$: To remove the existing party with identifier j from the group G using the keys of parties in set X. Once again, the oracle requires that $X \cap H \neq \emptyset$ because, otherwise, the adversary can revoke the member using

[4]This oracle requires that $X \cap H = \emptyset$. We note that this oracle, as well as $\mathsf{Delete2G}$ are meant to handle the cases where the adding and revoking parties are all corrupt. More on this can be found in Remark 2.

[5]More on this can be found in Remark 2.

```
$\mathsf{Exp}^{\mathsf{anonymity}}_{\sum,\mathcal{A},\lambda}$
$U = \emptyset, H = \emptyset, R = \emptyset, \mathsf{List} = \emptyset, \mathsf{Chall} = \perp, b \leftarrow \{0,1\}$
$(G, \Pi, \mathsf{state}) \leftarrow \mathcal{A}^{\mathsf{Add2U},\mathsf{Corrupt}}(1^\lambda)$
$\textbf{if } G \not\subseteq \mathcal{U}, \quad \textbf{return } \texttt{False}$
$(\mathsf{gvk}, \{\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i\}_{i \in G}) \leftarrow \sum.\texttt{KeyGen}(1^\lambda, G, \Pi)$
$\mathsf{state} = (\mathsf{gvk}, \{\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i\}_{i \in G \setminus H})$
$(m^*, X_0^*, X_1^*, \mathsf{state}) \leftarrow \mathcal{A}^{Oracles}(\mathsf{state})$
$\textbf{if } X_0^* \cap H = \emptyset \textbf{ or } X_1^* \cap H = \emptyset$
    $\textbf{or } X_0^* \cap R \neq \emptyset \textbf{ or } X_1^* \cap R \neq \emptyset \quad \textbf{return } \texttt{False}$
$\sigma^* \leftarrow \sum.\texttt{Sign}(m^*; \{\mathsf{gsk}_i\}_{i \in X_b^*})$
$\mathsf{Chall} = (m^*, \sigma^*)$
$b' \leftarrow \mathcal{A}^{Oracles}(\sigma^*, \mathsf{state})$
$\textbf{if } \exists\, X \subseteq (G \cup R) \setminus H : (X, \mathsf{open}) \in \Pi \quad \textbf{return } \texttt{False}$
$\textbf{elseif } (m^*, \cdot, X_0^*) \in \mathsf{List} \textbf{ or } (m^*, \cdot, X_1^*) \in \mathsf{List} \quad \textbf{return } \texttt{False}$
$\textbf{elseif } X_0^* \cap H = \emptyset \textbf{ or } X_1^* \cap H = \emptyset$
    $\textbf{or } (X_0^* \Delta X_1^*) \cap R \neq \emptyset \quad \textbf{return } \texttt{False}$
$\textbf{elseif } b' = b \quad \textbf{return } \texttt{True}$
$\textbf{else return } \texttt{False}$
```

Figure 7: Security game for anonymity. *Oracles* denotes the set of oracles the adversary has access to, specifically $\{\mathsf{Add2U}, \mathsf{Corrupt}, \mathsf{Add2G}, \mathsf{Append2G}, \mathsf{Sign}, \mathsf{SigOpen}, \mathsf{Remove2G}, \mathsf{Delete2G}\}$. We let Δ denote the symmetric difference operator for two sets (elements in exactly one of the two sets).

the keys it already has. The oracle runs $\sum.\texttt{RevokeMember}(j; \{\mathsf{rk}_i\}_{i \in X})$ and adds j to R.

- $\mathsf{Delete2G}(j, X)$: To remove the existing party with identifier j from the group G using the keys of parties in set X where $X \cap H = \emptyset$. The oracle removes j from G and adds j to R.

Game setup. In the beginning of each security game, the sets U, H, R and List are set to empty. In the anonymity game $\mathsf{Exp}^{\mathsf{anonymity}}_{\sum,\mathcal{A},\lambda}$, we additionally initialize the challenge signature C to $\perp$ and sample a random challenge bit b. $\mathcal{A}$ then gains access to the oracles $\mathsf{Add2U}$ and $\mathsf{Corrupt}$ using which it can add parties to the universe and corrupt some/all of them. $\mathcal{A}$ chooses a starting group G ($\subseteq U$), group structure Π and hands them over to $\mathcal{C}$. $\mathcal{C}$ runs $\sum.\texttt{KeyGen}(1^\lambda, G, \Pi)$ to obtain $(\mathsf{gvk}, \{\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i\}_{i \in G})$ and hands to $\mathcal{A}$ the $\mathsf{state} = (\mathsf{gvk}, \{\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i\}_{i \in G \setminus H})$.

Unforgeability game $\mathsf{Exp}^{\mathsf{forgery}}_{\sum,\mathcal{A},\lambda}$. In the unforgeability game, $\mathcal{A}$, with access

```
$\mathsf{Exp}^{\mathsf{revocability}}_{\sum,\mathcal{A},\lambda}$
$U = \emptyset, H = \emptyset, R = \emptyset, \mathsf{List} = \emptyset$
$(G, \Pi, \mathsf{state}) \leftarrow \mathcal{A}^{\mathsf{Add2U},\mathsf{Corrupt}}(1^\lambda)$
if $G \nsubseteq U$,  return False
$(\mathsf{gvk}, \{\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i\}_{i \in G}) \leftarrow \sum.\mathtt{KeyGen}(1^\lambda, G, \Pi)$
$\mathsf{state} = (\mathsf{gvk}, \{\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i\}_{i \in G \setminus H})$
$(m^*, \sigma^*, X^*) \leftarrow \mathcal{A}^{Oracles}(\mathsf{state})$
$Z \leftarrow \sum.\mathtt{SigOpen}(m^*, \sigma^*; \{\mathsf{ok}_i\}_{i \in X^*})$
if $\sum.\mathtt{Ver}(m^*, \sigma^*; \mathsf{gvk}) = 0$  return False
elseif $Z \nsubseteq G$  return True
else return False
```

Figure 8: Security game for revocability. *Oracles* denotes the set of oracles the adversary has access to, specifically $\{\mathsf{Add2U}, \mathsf{Corrupt}, \mathsf{Add2G}, \mathsf{Append2G}, \mathsf{Sign}, \mathsf{SigOpen}, \mathsf{Remove2G}, \mathsf{Delete2G}\}$.

to all the oracles described in Figure 4 and Figure 5, attempts to exhibit a valid (checked by $\sum.\mathtt{Ver}$) non-replayed (checked by List) message-signature pair that opens to a set containing an honest party. Since $\mathcal{A}$ did not have access to the keys of that honest party, this would count as a forgery. $\mathsf{Exp}^{\mathsf{forgery}}_{\sum,\mathcal{A},\lambda}$ outputs `True` if the adversary succeeds in exhibiting such a message-signature pair. Otherwise, $\mathsf{Exp}^{\mathsf{forgery}}_{\sum,\mathcal{A},\lambda}$ outputs `False`.

Anonymity game $\mathsf{Exp}^{\mathsf{anonymity}}_{\sum,\mathcal{A},\lambda}$. In the anonymity game, $\mathcal{A}$, with access to all the oracles described in Figure 4 and Figure 5, chooses a message and two sets of parties, each containing an honest party, and each containing no revoked parties (these checks are merely to ensure that both sets of parties can generate signatures but the adversary cannot generate signatures on behalf of either set of parties), and hands them over to $\mathcal{C}$. $\mathcal{C}$ then generates the challenge signature Chall, using the keys of one of the sets as determined by the challenge bit b that it chose at the beginning of the game. After receiving the challenge signature, $\mathcal{A}$, with continued access to all the oracles described in Figure 4 and Figure 5, tries to disambiguate the challenge signature. In doing so, $\mathcal{A}$ outputs a bit b', which if equal to b, constitutes a breach of anonymity and a win for $\mathcal{A}$. However, if $\mathcal{A}$ invoked oracles during the game that helped trivially disambiguate the challenge signature, then it does not win. This includes the following cases: (a) $\mathcal{A}$ has the keys corresponding to a set of parties with opening privileges, that is, $\mathcal{A}$ has the ability to open the challenge signature and trivially break anonymity; (b) $\mathcal{A}$ has effectively seen the challenge signature before (checked by List, more on this in Remark 5); (c) $\mathcal{A}$ has revoked parties such that verifying the challenge signature would reveal the set of parties used in its creation (that is, there is a revoked

```
$\mathsf{Exp}^{\mathsf{traceability}}_{\sum,\mathcal{A},\lambda}$
$U = \emptyset, H = \emptyset, R = \emptyset, \mathsf{List} = \emptyset$
$(G, \Pi, \mathsf{state}) \leftarrow \mathcal{A}^{\mathsf{Add2U},\mathsf{Corrupt}}(1^\lambda)$
**if** $G \not\subseteq U$,   **return** `False`
$(\mathsf{gvk}, \{\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i\}_{i \in G}) \leftarrow \sum.\mathtt{KeyGen}(1^\lambda, G, \Pi)$
$\mathsf{state} = (\mathsf{gvk}, \{\mathsf{gsk}_i, \mathsf{ik}_i, \mathsf{ok}_i, \mathsf{rk}_i\}_{i \in G \setminus H})$
$(m^*, \sigma^*, X^*, Y^*, r) \leftarrow \mathcal{A}^{Oracles}(\mathsf{state})$
$Z' \leftarrow \sum.\mathtt{SigOpen}(m^*, \sigma^*; \{\mathsf{ok}_i\}_{i \in X^*})$
$Z'' \leftarrow \sum.\mathtt{SigOpen}(m^*, \sigma^*; \{\mathsf{ok}_i\}_{i \in Y^*})$
$Z''' \leftarrow \sum.\mathtt{SigOpen}(m^*, \sum.\mathtt{Sign}(m^*; \{\mathsf{gsk}_i\}_{i \in X^*}, r); \{\mathsf{ok}_i\}_{i \in Y^*})$
**if** $\sum.\mathtt{Ver}(m^*, \sigma^*; \mathsf{gvk}) = 0$  **return** `False`
**elseif** $\{(m^*, \sigma^*, X^*), (m^*, \sigma^*, Y^*)\} \subset \mathsf{List}$  **return** `True`
**elseif** $Z' = \perp$ **or** $Z'' = \perp$ **or** $Z''' = \perp$  **return** `True`
**elseif** $Z' \neq Z''$  **return** `True`
**elseif** $(m^*, \sigma^*, \cdot) \in \mathsf{List}$ **and** $(m^*, \sigma^*, Z') \notin \mathsf{List}$  **return** `True`
**elseif** $Z''' \neq X^*$  **return** `True`
**else return** `False`
```

Figure 9: Security game for traceability. *Oracles* denotes the set of oracles the adversary has access to, specifically $\{\mathsf{Add2U}, \mathsf{Corrupt}, \mathsf{Add2G}, \mathsf{Append2G}, \mathsf{Sign}, \mathsf{SigOpen}, \mathsf{Remove2G}, \mathsf{Delete2G}\}$.

party that belongs to exactly one of the sets[6]). If none of these cases occurs and $\mathcal{A}$ did disambiguate the challenge signature successfully, this would count as a breach of anonymity and $\mathsf{Exp}^{\mathsf{anonymity}}_{\sum,\mathcal{A},\lambda}$ outputs `True`. Otherwise, $\mathsf{Exp}^{\mathsf{anonymity}}_{\sum,\mathcal{A},\lambda}$ outputs `False`.

Revocability game $\mathsf{Exp}^{\mathsf{revocability}}_{\sum,\mathcal{A},\lambda}$. In the revocability game, $\mathcal{A}$, with access to all the oracles described in Figure 4 and Figure 5, attempts to exhibit a valid (checked by $\sum$.`Ver`) message-signature pair that opens to a set containing a revoked party. (Technically, this is noted as $Z \not\subseteq G$ and not $Z \cap \mathcal{R} \neq \emptyset$. This is to cover the "impossible" cases of Z containing a party in $U \setminus (G \cup R)$). This would count as a breach of revocability and if the adversary succeeds in

[6]To be precise, this condition is more than what is required. All we need is that $\mathcal{A}$ never revoked a party that belonged to exactly one of the sets at a time when neither set contained a revoked party (and hence revoking this party would have implied different pre- and post-revocation behaviors for $\sum$.`Ver`). This can be captured in our game precisely by having a flag that would be updated by the oracles Remove2G and Delete2G. For ease of exposition, we choose to stick to this stronger condition that $\mathcal{A}$ has to satisfy, but note that the definition can be easily modified to support this and that our constructions do indeed satisfy this stronger notion of security.

exhibiting such a message-signature pair, $\mathsf{Exp}^{\mathsf{revocability}}_{\sum,\mathcal{A},\lambda}$ outputs `True`. Otherwise, $\mathsf{Exp}^{\mathsf{revocability}}_{\sum,\mathcal{A},\lambda}$ outputs `False`.

Traceability game $\mathsf{Exp}^{\mathsf{traceability}}_{\sum,\mathcal{A},\lambda}$. In the the traceability game, $\mathcal{A}$, with access to all the oracles described in Figure 4 and Figure 5, attempts to exhibit a valid (checked by $\sum$.`Ver`) message-signature pair that violates the traceability of the scheme. $\mathcal{A}$ may do so in the following ways: (a) $\mathcal{A}$ demonstrates a valid message-signature pair generated by the Sign oracle (checked by List) twice, with two different sets of creating parties; (b) $\mathcal{A}$ demonstrates a valid message-signature pair along with a set of opening parties that fails to open the signature; (c) $\mathcal{A}$ demonstrates a valid message-pair and two sets of opening parties that open the signature to two different sets of creating parties; (d) $\mathcal{A}$ demonstrates a valid message-signature pair that was generated by the Sign oracle (checked by List) with one set of creating parties, and a set of opening parties that opens the signature to another set of creating parties; and (e) $\mathcal{A}$ exhibits a message, a set of creating parties and the randomness (More on this in Remark 5) required to generate an "untraceable" signature–this signature when opened by a set of opening parties demonstrated by the adversary opens to a different set of creating parties. If any of these cases occurs, $\mathcal{A}$ did succeed in demonstrating a breach of traceability of the scheme and $\mathsf{Exp}^{\mathsf{traceability}}_{\sum,\mathcal{A},\lambda}$ outputs `True`. Otherwise, $\mathsf{Exp}^{\mathsf{traceability}}_{\sum,\mathcal{A},\lambda}$ outputs `False`.

Definition 1. *A managerless group signature scheme $\sum = \sum^{\mathsf{w/o\ manager}}_{\mathsf{Group}}$ is said to be secure if, for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$, the advantage of the adversary in the unforgeability, anonymity, revocability and traceability games, as defined below, are negligible in λ:*

$$\mathbf{Adv}^{\mathsf{forgery}}_{\sum,\mathcal{A},\lambda} = \Pr\left[\mathsf{Exp}^{\mathsf{forgery}}_{\sum,\mathcal{A},\lambda} = \mathit{True}\right]$$

$$\mathbf{Adv}^{\mathsf{anonymity}}_{\sum,\mathcal{A},\lambda} = \left|\Pr\left[\mathsf{Exp}^{\mathsf{anonymity}}_{\sum,\mathcal{A},\lambda} = \mathit{True}\right] - \frac{1}{2}\right|$$

$$\mathbf{Adv}^{\mathsf{revocability}}_{\sum,\mathcal{A},\lambda} = \Pr\left[\mathsf{Exp}^{\mathsf{revocability}}_{\sum,\mathcal{A},\lambda} = \mathit{True}\right]$$

$$\mathbf{Adv}^{\mathsf{traceability}}_{\sum,\mathcal{A},\lambda} = \Pr\left[\mathsf{Exp}^{\mathsf{traceability}}_{\sum,\mathcal{A},\lambda} = \mathit{True}\right]$$

3.3 Discussion

Noted below are a few observations given some of the choices underlying our security definition.

Remark 1. *Once the adversary corrupts a party, it can never "un-corrupt" it. Hence, once a party has been corrupted, we do not provide any security guarantees for that party.*

Remark 2. *We assume that all changes to* Table *are performed entirely honestly, that is, in accordance with the changes that are specified by the protocols*

`AddMember` *and* `RevokeMember`. *We stress that this is* not *limiting; this assumption is in accordance with our constructions as well as applications, where parties who wish to alter* Table *must generate a proof that shows that the changes made were indeed correct. Thus, when such changes do not involve any honest parties within our security game, we assume that the adversary communicates the change to the challenger by invoking the oracles* Append2G *and* Delete2G *right after making such changes. In the context of our application, edit access to* Table *would be restricted in a manner so as to ensure correct changes. Furthermore, for changes to G to take effect functionally in terms of generating signatures, etc., it is imperative that* Table *be updated accordingly, which would correspond in our security game to invoking the oracles* Append2G *and* Delete2G*. Additionally, when a new member is added to the group by corrupt parties alone, we require that the adversary honestly provide the keys of the added member to $\mathcal{C}$. We stress that this is* not *a limiting constraint and is merely for book-keeping and ease of exposition. An underlying assumption with regards to this, however, is that any party picks their keys at the times of their addition to the group and that they use these keys while performing subsequent operations. This can and will be enforced by the integrity of* Table *in our constructions.*

Remark 3. *We do not allow the adversary to add a revoked member back into the group again. This is merely to simplify the exposition. With some more book-keeping, it is possible to allow the adversary to add revoked members back into the group, without any change to security guarantees that are already offered.*

Remark 4. *We do not consider it a security breach if revoked members open signatures that were generated after they had been revoked. We discuss this in more detail while describing our construction of a managerless group signature scheme, but at a high-level, we note that if this were not the case, then, when a member is revoked, other members would necessarily have to interact and refresh their keys. It is possible to include such a protocol in the syntax of the scheme and then one could provide a stronger guarantee that revoked members lose their opening privileges. For efficiency concerns, we omit such a re-keying protocol from the syntax of our scheme and hence also allow for revoked members to continue opening signatures. While revoked members may still attempt to add new members by interacting with them and helping them generate their own new (potentially valid) keys, they will never be able to successfully add new members. The reason is that revoked members as well as users who interacted with a revoked (at the time) member in an attempt to be added to the group will not be able to generate signatures that verify. This will be ensured because of* Table *being refreshed when a member is revoked. Additionally, such members will not be able to revoke other members as well, since revocation would involve refreshing* Table *in a verifiable way, where the identities of the parties performing the revocation is public. Thus, informally, revoked members as well as users who interacted with a revoked (at the time) member in an attempt to be added to the group will be able to open signatures but not add or revoke other members. Note that in a sense, users who interacted with a revoked (at the time) member in an attempt to be added to the group are functionally similar to revoked members* *themselves.*

Remark 5. *In our security definition, we implicitly account for the signature scheme's signing algorithm possibly being deterministic. That is, it would be possible for such a scheme to still be secure under our definition. This is done through a myriad of checks sprinkled throughout the game, for example, the adversary is not allowed to see a signature on a message created by some set of parties and then challenge anonymity with that very message and set of parties; if signing were deterministic, an adversary who is allowed to make such challenges can trivially break anonymity. It is thus important to note in this context that if one had additional guarantees regarding the signature scheme, for instance that signing is "well-randomized", one would potentially be able to come up with a game where the adversary has additional capabilities and thus put forth a definition that provides stronger security guarantees. We however do not go into these enhancements for the purposes of our work.*

4 Building MGS for FIDO

4.1 Supporting Unilateral Adds

A näive approach to support a group of signers would be to simply share the secret key with all members of the group. However, sharing secret keys with additional parties is something which the FIDO and WebAuthn standards discourage. See for example the definition[7] of a "credential private key" where it states the "private key . . . is expected to never be exposed to any other party". Instead we will support groups where each member holds a different secret key but these are all verified by the same public key. One of the important design considerations is the group structure we would like to support. In this work, we will always support *unilateral signing* of messages, that is, any single current group member can sign messages on their own. In our first construction, we would like to support unilateral adding of members, that is, any single current member of the group can add a new member on their own. This group structure has some consequences, which we also discuss.

A simple way to construct a group signature scheme where any single existing member of the group can unilaterally add a new member is as follows. Consider a regular signature scheme where all members of the group will possess the secret signing key. Clearly, any member of the group can unilaterally add a new member by giving them the signing key as well. While this is technically correct, for applications which might use such a primitive, an added restriction is that parties may not share keys.[8] We show that by using signing key re-randomizable signature schemes, we can make the previous idea work. At a high-level, while added a new member, the existing member does not share their own key, but rather "re-randomizes" their key to provide a new key for the new member.

[7] `https://www.w3.org/TR/webauthn-2/#credential-private-key`

[8] Alternatively, secret keys may not leave the device.

Formally, let $\sum_{\mathsf{SR}}$ be a signing key re-randomizable signature scheme with protocol Π_{ReRand} for $\mathcal{F}_{\mathbf{ReRand}}$. We define a group signature scheme $\sum_{\mathsf{Group,unilateral+}}^{\mathsf{w/o\ manager}}$ for a group G with the unilateral-add group structure consisting of $|G| = n$ members without a group manager to consist of the following protocols:

- $\mathtt{KeyGen}(1^\lambda, G)$: Samples $(\mathsf{vk}, \mathsf{sk}) \leftarrow \sum_{\mathsf{SR}}.\mathtt{KeyGen}(1^\lambda)$ and sets $\mathsf{gvk} = \mathsf{vk}$. Then, for each $i \in [|G|]$, the algorithm runs Π_{ReRand} (playing the role of both parties) for the scheme $\sum_{\mathsf{SR}}$ using input sk to obtain ik_i. Set $\mathsf{gsk}_i = \mathsf{ik}_i$.

- $\mathtt{AddMember}(j, \mathsf{ik}_i)$: Parties i and j run Π_{ReRand} using ik_i as i's input with j obtaining $\mathsf{gsk}_j = \mathsf{ik}_j$.

- $\mathtt{Sign}(m; \mathsf{gsk}_i)$: Generates $\sigma \leftarrow \sum_{\mathsf{SR}}.\mathtt{Sign}(m; \mathsf{gsk}_i)$.

- $\mathtt{Ver}(m, \sigma; \mathsf{gvk} = \mathsf{vk})$: Outputs $\sum_{\mathsf{SR}}.\mathtt{Ver}(m, \sigma; \mathsf{vk})$.

Notice that the above scheme is fully-anonymous. In fact, it is not possible to learn the identity of the signer of a message from the signature. Indeed, signatures created by all members are identically distributed as they are simply signatures from a single regular signature scheme. Thus, revocation, in this context, would essentially necessitate re-keying the entire scheme. This is the reason for omitting ok_i, rk_i, $\mathtt{SigOpen}$ and $\mathtt{RevokeMember}$ from the description. Owing to these syntactic differences and the stronger anonymity we achieve in this case, we make the following formal claim regarding the anonymity of our construction (rather than working with the definition of anonymity in Definition 1).

It is easy to see that the anonymity guarantee provided by Lemma 1 is at least as strong as that guaranteed in Definition 1 as the adversary's capability in Lemma 1 is strictly stronger than in Definition 1 (barring the ability to revoke members and open signatures) as the adversary has access to keys of not just the corrupt parties but also the honest parties.

Lemma 1. *Let $\sum_1 = \sum_{\mathsf{Group,unilateral+}}^{\mathsf{w/o\ manager}}$ be the group signature scheme defined above. Assuming $\sum_{\mathsf{SR}}$ is a secure signing key re-randomizable signature scheme, for any adversary $\mathcal{A}$, the quantity $\mathbf{Adv}_{\sum_1,\mathcal{A},\lambda}^{\mathsf{strong-anonymity}}$ is $\frac{1}{2}$, where $\mathbf{Adv}_{\sum_1,\mathcal{A},\lambda}^{\mathsf{strong-anonymity}}$*

is defined to be

$$\Pr\left[\begin{array}{c} G \leftarrow \mathcal{A} \\ (\mathsf{gvk}, \{\mathsf{gsk}_i, \mathsf{ik}_i\}_{i\in[|G|]}) \leftarrow \sum_1.\mathtt{KeyGen}(1^\lambda, G) \\ \mathcal{Q} = \{\mathsf{gvk}\} \cup \{\mathsf{gsk}_i, \mathsf{ik}_i\}_{i\in[|G|]} \\ \left\{\begin{array}{c} \mathsf{ik}_j \leftarrow \mathcal{A}(\mathcal{Q}) \\ (\mathsf{gsk}, \mathsf{ik}) \leftarrow \sum_1.\mathtt{AddMember}(\mathsf{ik}_j) \\ \mathcal{Q} = \mathcal{Q} \cup \{\mathsf{gsk}, \mathsf{ik}\} \end{array}\right\}_j \\ m \leftarrow \mathcal{A}(\mathcal{Q}) \\ I_0, I_1 \leftarrow [|G|] : I_0 \neq I_1,\ b \leftarrow \{0,1\} \\ \sigma \leftarrow \sum_1.\mathtt{Sign}(m; \mathsf{gsk}_{I_b}) \\ \left\{\begin{array}{c} \mathsf{ik}_j \leftarrow \mathcal{A}(\mathcal{Q}) \\ (\mathsf{gsk}, \mathsf{ik}) \leftarrow \sum_1.\mathtt{AddMember}(\mathsf{ik}_j) \\ \mathcal{Q} = \mathcal{Q} \cup \{\mathsf{gsk}, \mathsf{ik}\} \end{array}\right\}_j \\ b' \leftarrow \mathcal{A}(Q, I_0, I_1, m, \sigma) \end{array} : b = b'\right]$$

Proof. The proof of the lemma is rather straightforward. Indeed, the entire view of $\mathcal{A}$, prior to the sampling of I_0, I_1 and b, reduces to the key-pair $(\mathsf{vk}, \mathsf{sk})$ sampled during $\sum_1.\mathtt{KeyGen}$ as $(\mathsf{vk}, \mathsf{sk}) \leftarrow \sum_{\mathsf{SR}}.\mathtt{KeyGen}(1^\lambda)$, where $\sum_{\mathsf{SR}}$ is a signing key re-randomizable signature scheme. Furthermore, it is also straightforward to observe that for m chosen by the adversary, the distributions $\{\sum_1.\mathtt{Sign}(m; \mathsf{gsk}_{I_0})\}$, $\{\sum_1.\mathtt{Sign}(m; \mathsf{gsk}_{I_1})\}$ and $\{\sum_{\mathsf{SR}}.\mathtt{Sign}(m; \mathsf{sk})\}$ are identical. Hence, the probability that $\mathcal{A}$ guesses $b' = b$ is exactly $\frac{1}{2}$. $\square$

5 Applying MGS to FIDO

5.1 Overview of FIDO User Authentication

FIDO's Universal Authentication Framework (UAF) protocol [21] uses public-key cryptography in conjunction with a local authentication such as PIN, password, or biometrics on a user device such as a smartphone or laptop (FIDO UAF authenticator) for remote user authentication by a relying party server (FIDO server). Another FIDO protocol called the Universal 2nd Factor (U2F) [20] allows a relying party server to request a second factor for stronger two-factor authentication using the same FIDO challenge-response cryptographic protocol. Later on, FIDO alliance worked jointly with the Worldwide Web Consortium (W3C) to introduce the second generation of FIDO protocols commonly called FIDO2. Figure 12 in Appendix A shows the registration and authentication flows.

Registration. When a user intends to register a FIDO authenticator associated with an account with a relying party FIDO server, the authenticator generates a unique public-private key pair, corresponding to the user account and associated relying-party service. The private key is stored securely on the authenticator and the public key is registered with the server.

Authentication. When a relying-party service intends to authenticate the user, the FIDO server sends a challenge to the FIDO authenticator through the client. This challenge prompts the user to use a registered FIDO authenticator. Local user authentication unlocks the corresponding private key stored on the authenticator which is used to sign the server's challenge. The signed challenge is sent to the server and verified using the corresponding registered public key for user authentication.

5.2 Enrollment & Authentication using MGS

The proposed group signature protocol can be used for user authentication, either as a passwordless or second-factor authentication mechanism. Similar to FIDO authentication, we consider three main entities: 1) a client machine C on which the user initiates a login session. 2) a relying party server S enrolling and authenticating users, and 3) Group of Devices $D_1, \ldots, D_n$ serving as external (roaming) authenticators. We assume that all these entities run an application supporting our protocol, e.g., an app on a smartphone and browser extension on the client. The interaction between the server and device is handled through the client. Specifically, the client interacts with the server to receive a challenge message and passes it to a subset of devices in the group, who will sign the challenge as evidence that registered devices are available during authentication. The client receives the response from the devices and sends it to the server for verification.

Setup: setup runs among the parties to establish a secure connection. This setup can be run prior to or during new device registration and consists of the following processes:

- Setup$_D$: the setup process running between devices to discover each other and run a key exchange to secure their connection.
- Setup$_{D-C}$: the setup process running between device(s) and the client to establish a secure connection.
- Setup$_{C-S}$: trivially the client to server connection is secured using common techniques such as TLS over the internet.

While the setup process is out of the scope of our protocol, we can consider that this procedure is performed by the user. For example, the user connects the client and their devices to the same network (or turns on Bluetooth), installs the client browser extension, installs the app on the devices, connects devices with each other and pairs devices with the client. In FIDO, each authenticator is registered independently with the relying party and hence a connection between devices is not needed. Therefore, Setup$_D$ is specific to our protocol, while other parts of the setup process are implicitly required by FIDO, too. Similar to our approach, the Yubico backup and recovery proposal [23] assumes a communication channel between authenticators.

Registration (adding device(s) for the first time): registration (a.k.a. enrollment) is done using the `KeyGen` protocol of the managerless group signature scheme. It runs between the new device(s) and the server to sign up the first authenticator(s) with the authentication protocol offered by the server. The $(\mathsf{gsk}_i, \mathsf{gvk})$ key pair is generated on device i using the `KeyGen` protocol of the managerless group signature scheme. The device stores gsk_i securely and gvk is transferred to the server. The list Table that is part of the managerless group signature scheme is created and maintained on the server. As part of the `KeyGen` protocol, a new entry would be added to Table which can be referenced by a device nickname. At the end of this phase, a round of authentication can take place to verify the enrollment. This stage is similar to registering to FIDO authentication with a server and adding a FIDO-enabled security key. The main difference is the type of the key-pair generated on the device.

The user interaction flow can remain similar to FIDO authentication enrollment. The user logs in to the server using the first authentication factor (e.g., username and/or password) to enroll in the service. The server sends a request to the client for it to obtain and pass the group public key gvk to the server. The client forwards the message to all connected devices known through the Setup_{D-C}. The user receives a notification on all devices but may approve the registration on only one of the devices (lets say D_i which consequently runs the `KeyGen` protocol with all other connected devices known through Setup_D. At this stage the user may also get the option to form a group of devices she wishes to correspond to the enrolling service/account. For example, the user may select only her work laptop and work smartphone when registering to a FIDO service to access workplace services. Hence, only the two devices run the `KeyGen` protocol. At the end of this round, the gvk and a list of all members of the group are passed to the server through the client and updated on the client and the server. Parties can run one authentication attempt to validate the protocol.

Authentication: The authentication flow is similar to FIDO user authentication (Figures 1(b) and 12(b)). The server sends a one-time nonce through the client to the registered devices. The user can attend (or locally authenticate to) any of the available devices, which sign the challenge (the nonce and any input from the client), using the `Sign` protocol of the managerless group signature scheme. The response is returned to the server and the signature is verified using the `Ver` protocol of the managerless group signature scheme.

Client Sync: Authentication from a new client does not require re-enrollment. However, the setup should happen between devices and the new client.

Add extra device(s): After the initial registration of the first authenticator(s), adding extra devices only relies on the `AddMember` protool of the managerless group signature scheme. To add additional devices the user first runs a setup processes to establish connections to the new device. On any of the previously enrolled devices, the user selects services that the new device should be enrolled with and runs the `AddMember` protocol to add the new group member. Depending on the group structure that the user wants, this step would involve one or more registered authenticator(s) to participate in addition to the new authenticator. The system can update the device list (Table) on the server during the first authentication attempt from the new device. Unlike the enrollment of the initial devices, adding a new member can run only between the user's devices and does not require the devices to interact with the server. However, to keep the user flow similar to FIDO, the user can initiate the process on the client (Figure 1(a) and 12(a)).

Revocation: In case one of the devices gets lost, the user can run the `RevokeMember` protocol of the managerless group signature scheme between (some subset of) registered devices to remove the lost device from the group and update the list (Table) on the server accordingly after the revocation (either immediately or during the next authentication call). Similar to the FIDO protocol, such revocation is important to make sure that the lost device does not take part in future login attempts, particularly when local authentication to the authenticator is not in place.

Quorum-based addition and revocation: As discussed in Section 7.1, when adding or revoking members, we can also use the threshold version of our managerless group signature scheme, thereby requiring a threshold number (a tunable parameter) of devices to participate in the add or revoke member protocols and provide protection against maliciously added devices gaining control of the group.

Multiple Relying Parties: A major advantage of our approach is that it streamlines the enrollment of backup devices across relying parties. While other approaches require enrollment of a backup device/authenticator with each relying party account of a user, we instead can run multiple instances of the group signature protocol among the backup devices. When enrolling a new backup device, we run the registration for each protocol instance concurrently (without any need to contact the individual relying parties).

5.3 Security in FIDO

The FIDO Security Reference [19] defines the exact security properties required in the FIDO setting. The two properties relevant in the context of this work are Unforgeability [19, cf. SG11] and Unlinkability [19, cf. SG4]. Earlier work by Barbosa et al. [8] performed a detailed analysis of the FIDO 2.0 protocol focusing on unforgeability and we can reuse some of their results here.

Security Model. First let us recap the model and assumptions made by Barbosa et al. [8]. In their analysis they consider WebAuthN as a Passwordless Authentication protocol (PlA), between three parties: an authenticator (held by the user - referred to by Barbosa et al. as a token), a client (such as web browser), and a relying party server. We make no assumptions about the communication channels between parties. Authenticators are assumed to be tamper-proof, and have an attestation keypair injected at manufacture with a certified root pre-registered with relying parties.

The PlA protocol will perform two operations: enrollment (registration) and authentication. At a high-level, a PlA is secure (unforgeable) if, when a relying party successfully completes the authentication protocol, there exists a unique authenticator that has engaged in the same session of the protocol.

Unforgeability. The following result by Barbosa et al. [8], showed that the WebAuthN protocol is a secure PlA protocol given the underlying signature scheme is unforgeable.

Theorem (Theorem 1 of [8] restated)**.** *Assuming H is a collision resistant hash function, and the signature scheme* sig *is* existentially unforgeable*, then WebAuthn is a secure PlA protocol.*

The current Webauthn protocol is instantiated using RSASSAPKCS1-v1 5 and RSASSA-PSS which we know to be existentially unforgeable in the random oracle model. This result also shows that we could securely use WebauthN with our new group signature scheme since it is also unforgeable (Theorem 1).

Unlinkability. The FIDO Security Reference defines unlinkability as the ability to *protect the protocol conversation such that any two relying parties cannot link the conversation to one user (i.e. be unlinkable).* Prior work on analyzing this property from a symbolic perspective has been performed by Feng et al. [18].

Assuming the same security model as above we can make more straightforward arguments (without a mathematical security definition) about the unlinkability of our scheme:

1. As each group generates a unique group verification key for each relying party. This is equivalent to the standard FIDO approach where a unique key pair is generated when enrolling with new relying parties as analyzed by Feng et al. [18]. Hence linking across parties based on public keys is not possible.

2. Moreover, in the full version of the paper we show that our MGS scheme additionally achieves anonymity properties and hence it is not possible to determine which member of the group signed a message. It is therefore impossible to link signed messages according to their creator. While this is not required by FIDO it is still a desirable property to achieve.

6 Implementation and Evaluation

6.1 System Implementation and Setup

To study the feasibility of the system and understand its performance, we deployed a proof of concept implementation of the system. We developed a web server application that verifies the username and password as the first factor and runs our managerless group signature protocol to validate the presence of a registered device. Following applications and services were developed as part of the proof of concept.

- The managerless group signature library to generate signing key pairs, add a new member to a group, sign a message, and verify a signature. We implemented our protocol $\sum_{\mathsf{Group,unilateral+}}^{\mathsf{w/o\ manager}}$ from Section 4.1 that supports unilateral signing and unilateral addition of members. We implement two versions of our scheme: one based on Schnorr [39] and the other based on ElGamal [24] signature schemes as the underlying signing-key rerandomizable signature schemes. We used the following libraries as part of this implementation: BouncyCastle [13], and SpongyCastle [41].

- An Android app developed in Java for Android that can generate a signature, add new members, and being added as a member to a group using our group signature library. The app receives a challenge from the client, signs the challenge, and returns the response to the client.
- A Chrome browser extension developed in Javascript that communicates with the web server to receive a nonce, transfer it to the device, receive the response from the device, and relay it to the web server.
- An authentication server developed in Java for generating a challenge and verifying the response. Signature verification uses our group signature library.
- A web server running PHP and Javascript scripts to interact with the user (receiving username and password), to communicate with the authentication server to receive the nonce and send the response, to communicate with the browser extension to send the nonce and receive the response. The webserver and the authentication server can be considered as one entity, namely the relying party.
- Communication between the web server and server happens over the internet using UDP sockets, where the authentication server runs the UDP server and the web server runs the UDP client part. Communication between the client and the web server is facilitated using Chrome messaging API. A client-server UDP communication runs between the devices and Chrome browser extension.

6.2 Performance Evaluation

We implement our group signature schemes (both using Schnorr and ElGamal) using the standard P-256 256-bit prime field Weierstrass elliptic curve (also known as secp256r1 and prime256v1). We report the execution and communication cost of our protocol, averaged over 10000 iterations, in Table 1. In this evaluation, all entities are connected to the same WiFi local network. The relying party is executed on a HP Envy x360 laptop with Intel Core i7-6500U processor and 16GB memory running Ubuntu 20 operating system. The client machine is a Macbook Air with 1.3GHz Intel Core i5 and 4GB of memory. We executed the android app on a Huawei Honor 7X with Kirin 659 processor and 3GB memory. As shown in Table 1 and 2, the cryptographic operations are lightweight and the total execution cost for one round of authentication on a local network is about 100ms. We compare our performance with that of the ECDSA signature scheme [30] using NIST P-256 curve, which is suggested as part of the FIDO2 specification. Our ECDSA implementation uses Java Security Library and follows the example given in [35].

Table 1: Average communication and running time of cryptographic operations used in the managerless group signature scheme vs that of the ECDSA scheme used in FIDO2. Time is shown in milliseconds; n is the number of public keys registered for a user account (i.e number of authenticators).

Signature	**El Gamal**	**Schnorr**	**ECDSA**
Curve Type	secp256r1		
Key Gen	3.12	3.26	1.58
Signing	3.07	3.18	2.23
Verification	6.20	4.93	n x 2.22
Add Member	0		1.58
Communication	≈ 1 ms		

Inferences. We observe that the cost of our key generation and signing algorithms, is comparable to ECDSA. Our verification algorithm has a fixed cost since there is only a single group public key, while in the case of ECDSA, each authenticator has a different public key and hence, the verification needs to be processed with each of them. The biggest advantage of our scheme is in the cost of adding a new member – it is almost negligible since it only involves re-randomizing an existing key. However, in the ECDSA version used currently in FIDO2, adding a member requires running the key generation algorithm again. For communication, as part of key generation, the user sends one message from the new device to the server. When adding a member, the old member sends one message to the new one being added. The signing and verification are non-interactive.

Given the low cryptographic operation costs that are comparable to ECDSA (and similar to that of the current FIDO specification), the protocol works well even on resource constrained devices. In this evaluation, we did not include the cost of system setup (not the key generation) as it happens only once.

With respect to user interaction model, user authentication flow is similar in our method and FIDO2 as shown in Figure 1(b) and 12(b). In both case, the user starts the login from the client machine, and approves the login on any of the authenticators, for example by using FaceID to locally authenticate to the smartphone. The key difference between our technique and FIDO2 is registration of a new authenticator, as shown in Figure 1(a) and 12(a). While in FIDO2, each authenticator is enrolled independently with the relying part, in our technique, the two authenticators communicate to add the new authenticator as to the group. From the user's interaction perspective this process can be similar to FIDO2 registration. Of course, we need to assume a communication channel has been setup between the two authenticators. Note that the communication channels are often in place specially when authenticators are users' personal devices such as laptops and smartphones. Hence, the usability of the proposed system is similar to FIDO2 and has previously been studied in depth (e.g., [14, 32, 15]).

Table 2: Group operations in the managerless group signature scheme vs that of the ECDSA scheme used in FIDO2. Mul denotes multiplication and Exp denotes exponentiation; n is same as in the previous table.

Signature	**Schnorr**		**El Gamal**		**ECDSA**	
	Mul	**Exp**	**Mul**	**Exp**	**Mul**	**Exp**
Key Gen	0	1	0	1	0	1
Signing	0	1	0	1	0	1
Verification	1	2	1	3	n	$1+n$
Add Member	0	0	0	0	0	1

7 MGS with Full Security

The unilateral adds scheme from Section 4.1, while being fully anonymous and sufficient for the FIDO context, is not traceable at all. This additional notion is not needed for FIDO but is of theoretical interest when construct the best possible MGS scheme. One could now ask if not achieve it is a drawback of our first scheme or a consequence of the group structure itself. To better understand the situation, we posit that there is a way for parties to uniquely identify themselves.[9] For our scheme to be traceable, it should be possible to "open" a signature and determine the identity of its signer. At a high level, since a single member was able to unilaterally add a new member, an adversarial member could unilaterally "add" an old member once again. That is, an adversarial member can "manufacture" valid credentials, unless there is some *public information that hides and yet authenticates the credentials of members.* Essentially, when a member is added, the party adding them needs to generate some public certificate that hides the new member and yet allows it to authenticate itself with respect to this certificate. An underlying assumption is that parties cannot alter this public information arbitrarily, but only do so under some moderation. Therefore, an adversary cannot re-write this information in an attempt to generate some other certificate for another party. Since the new member alone will be able to prove knowledge of the secret information that was agreed upon when they joined the group (that is, authenticate itself with respect to this certificate), an adversary cannot re-use this public certificate on behalf of any other party. To avoid timing attacks, as well as members being added without their consent, there must be a mechanism for parties to explicitly prove their interest in joining the group. With these assumptions in place, using ideas from [9], we can extend our scheme from the previous section to also be traceable. Further discussion on how our extended scheme captures traceability

[9]We can assume that there is some sort of identity platform in place. This is done to avoid the case of an adversary simply creating spurious "ghost" parties.

can be found in the full version of the paper.

We prove the security of our construction in Figure 11 in Appendix F. For completeness, we present the formal theorem statement proving security here.

Theorem (Theorem 1 from Appendix F). *Assuming* Com *is a computationally hiding and perfectly binding commitment scheme,* $\sum_{\mathsf{SR}}$ *is a secure signing key re-randomizable signature scheme,* Γ *is a CCA2-secure public key encryption scheme and* NIZK *is a a non-interactive zero-knowledge argument, scheme* $\sum_{\mathsf{Group}}^{\mathsf{w/o\ manager}}$ *(Figure 11) is a dynamic managerless group signature scheme that satisfies Definition 1 with respect to group structure* Π_{uni}.

7.1 Further Extensions

Forward Security. As in the case of [9], we observe that our group signature scheme achieves forward security if the underlying signing key re-randomizable signature scheme $\sum_{\mathsf{SR}}$ is forward secure. That is, if $\sum_{\mathsf{SR}}$ is forward secure, any adversary $\mathcal{A}$ would not be able to produce valid signatures for an earlier time period given only keys corresponding to a later one. Also, as noted earlier, achieving signing key re-randomizability is straightforward in most signature schemes, including forward secure ones.
Supporting Threshold Group Structures. In order to support threshold group structures, we simply distribute all keys in an appropriate threshold manner. Note that simple Shamir secret sharing suffices. Furthermore, Shamir secret sharing allows us to generate fresh shares, given the secret, which can be done using a simple multi-party computation protocol.[10] This would be of importance while adding or revoking new members.
Symmetry of Group Members. A final note we present is regarding the possibility of the group structure being asymmetric towards group members. For instance, this may also mean that not all parties have the same signing, adding, opening and/or revoking capabilities. It is important to note that different (possibly asymmetric) group structures may allow for more efficient schemes, involving fewer or more efficient instances of multi-party computation protocols.

8 Conclusion

We propose a method that inherently enables multiple FIDO authenticators per user account for account recovery. The proposed method organizes FIDO authenticators into groups and assigns each authenticator in a group a unique private key for an account with a relying party. Each authenticator in a group can sign relying party challenges independently using its own private key. However, the relying party server can use the same verification key to validate challenges

[10]This would be a simple polynomial interpolation, but there may be other, more efficient options involving threshold cryptography and/or other threshold sharing schemes.

$\sum_{\mathsf{Group}}^{\mathsf{w/o\ manager}}$

Technical Ingredients: A commitment scheme Com, signing key re-randomizable signature scheme $\sum_{\mathsf{SR}}$ with protocol Π_{ReRand} for $\mathcal{F}_{\mathbf{ReRand}}$ (with associated simulator $\mathsf{Sim}_{\mathbf{ReRand}}$), a CCA2-secure public key encryption scheme Γ and a non-interactive zero-knowledge argument NIZK.

<u>$\texttt{KeyGen}(1^\lambda, G, \Pi_{\mathsf{uni}})$:</u>

- Sample $\mathsf{pp} \leftarrow \mathsf{Com}.\texttt{Gen}(1^\lambda)$, $\mathsf{crs} \leftarrow \mathsf{NIZK}.\texttt{CRSGen}(1^\lambda)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \sum_{\mathsf{SR}}.\texttt{KeyGen}(1^\lambda)$ and $(\mathsf{ek}, \mathsf{dk}) \leftarrow \Gamma.\texttt{KeyGen}(1^\lambda)$.
- Sample random values r, ω and commitment $\mathsf{com} = \mathsf{Com}(r; \omega)$. Set $\mathsf{gvk} = (\mathsf{pp}, \mathsf{crs}, \mathsf{vk}, \mathsf{ek}, \mathsf{com}, \mathsf{Table})$ where $\mathsf{Table} = \perp$.
- For each $i \in [|G|]$,
  - Run Π_{ReRand} (playing the role of both parties) using input sk to obtain sk_i.
  - Compute commitments $\mathsf{com}_{i,1} = \mathsf{Com}(r_i; \omega_i)$, $\mathsf{com}_{i,2} = \mathsf{Com}(\alpha_i; \cdot)$ using randomness r_i, ω_i, α_i.
  - Compute ciphertext $\mathsf{ct}_i = \Gamma.\texttt{Enc}((\mathsf{com}_{i,1}, \mathsf{com}_{i,2}); \mathsf{ek}, \beta_i)$ using randomness β_i.
  - Set $\mathsf{gsk}_i = (\mathsf{sk}_i, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i)$, $\mathsf{ik}_i = (\mathsf{sk}_i, \mathsf{dk}, r, \omega)$, $\mathsf{ok}_i = \mathsf{dk}$ and $\mathsf{rk}_i = (r, \omega, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i)$.
  - Set $\mathsf{Table}_i = (\mathsf{ct}_i, \texttt{INIT}, \texttt{IN})$.
  - Add $(\{i\}, \mathsf{add})$, $(\{i\}, \mathsf{sign})$, $(\{i\}, \mathsf{open})$ and $(\{i\}, \mathsf{revoke})$ to Π.

<u>$\texttt{AddMember}(j; \mathsf{ik}_i = (\mathsf{sk}_i, \mathsf{dk}, r, \omega))$:</u>

- Parties i and j run Π_{ReRand} using sk_i as i's input with j obtaining sk_j. Party i also passes along (dk, r, ω) to party j.
- Party j computes commitment $\mathsf{com}_{j,1} = \mathsf{Com}(r_j; \omega_j)$ using random values r_j, ω_j.
- Party i computes commitment $\mathsf{com}_{j,2} = \mathsf{Com}(\alpha_j; r_{j,2})$ using random values $\alpha_j, r_{j,2}$.
- Parties i and j then exchange $\mathsf{com}_{j,1}$ and $\mathsf{com}_{j,2}$.
- Party j computes ciphertext $\mathsf{ct}_j = \Gamma.\texttt{Enc}((\mathsf{com}_{j,1}, \mathsf{com}_{j,2}); \mathsf{ek}, \beta_j)$ using randomness β_j. Sends (ct_j, β_j) to party i.
- Party i computes $\pi \leftarrow \mathsf{NIZK}.\texttt{Prove}(\mathsf{stmt}_{\mathsf{add}}; [r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i, \mathsf{com}_{j,1}, \alpha_j, r_{j,2}, \beta_j])$ where $\mathsf{stmt}_{\mathsf{add}} = (i, \mathsf{ct}_j, \mathsf{gvk})$ and the relation is defined by: $\exists\ [r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i, \mathsf{com}_{j,1}, \alpha_j, r_{j,2}, \beta_j]$: $\mathsf{Table}_i = (\Gamma.\texttt{Enc}((\mathsf{Com}(r_i; \omega_i), \mathsf{com}_{i,2}); \mathsf{ek}, \beta_i), \cdot, \texttt{IN})$ and $\mathsf{ct}_j = \Gamma.\texttt{Enc}((\mathsf{com}_{j,1}, \mathsf{Com}(\alpha_j; r_{j,2})); \mathsf{ek}, \beta_j)$ and sets $\mathsf{Table}_j = (\mathsf{ct}_j, (i, \pi), \texttt{IN})$.
- Party i adds j to G and $(\{j\}, \mathsf{add})$, $(\{j\}, \mathsf{sign})$, $(\{j\}, \mathsf{open})$ and $(\{j\}, \mathsf{revoke})$ to Π.
- Finally, party j sets $\mathsf{gsk}_j = (\mathsf{sk}_j, r_j, \omega_j, \mathsf{com}_{j,2}, \beta_j)$, $\mathsf{ik}_j = (\mathsf{sk}_j, \mathsf{dk}, r, \omega)$, $\mathsf{ok}_j = \mathsf{dk}$ and $\mathsf{rk}_j = (r, \omega, r_j, \omega_j, \mathsf{com}_{j,2}, \beta_j)$.

Figure 10: Our dynamic managerless group signature scheme $\sum_{\mathsf{Group}}^{\mathsf{w/o\ manager}}$ (Part 1/2).

$\sum_{\mathsf{Group}}^{\mathsf{w/o\ manager}}$

$\underline{\mathtt{Sign}(m; \mathsf{gsk}_i = (\mathsf{sk}_i, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i))}$:

- Generate signature $\sigma' \leftarrow \sum_{\mathsf{SR}}.\mathtt{Sign}(m; \mathsf{sk}_i)$. Compute ciphertext $\mathsf{ct} = \Gamma.\mathtt{Enc}([i, \sigma']; \mathsf{ek}, r)$ using randomness r.
- Compute proof $\pi \leftarrow \mathsf{NIZK}.\mathtt{Prove}(\mathsf{stmt}_{\mathsf{sign}}; [i, \sigma', r, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i])$ where $\mathsf{stmt}_{\mathsf{sign}} = (m, \mathsf{ct}, \mathsf{gvk})$ and the relation is defined by: $\exists\ [i, \sigma', r, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i]$: $\sum_{\mathsf{SR}}.\mathtt{Ver}(m, \sigma'; \mathsf{vk}) = 1$, $\mathsf{ct} = \Gamma.\mathtt{Enc}([i, \sigma']; \mathsf{ek}, r)$ and $\mathsf{Table}_i = (\Gamma.\mathtt{Enc}((\mathsf{Com}(r_i; \omega_i), \mathsf{com}_{i,2}); \mathsf{ek}, \beta_i), \cdot, \mathtt{IN})$.
- Set $\sigma = (\mathsf{ct}, \pi)$.

$\underline{\mathtt{SigOpen}(m, \sigma = (\mathsf{ct}, \pi); \mathsf{ok}_i = \mathsf{dk})}$:
Output $\perp$ if $\mathtt{Ver}(m, \sigma; \mathsf{gvk}) = 0$. Else, compute plaintext $[i, \sigma'] = \Gamma.\mathtt{Dec}(\mathsf{ct}; \mathsf{dk})$ and output i.

$\underline{\mathtt{RevokeMember}(j; \mathsf{rk}_i = (r, \omega, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i))}$:

- Compute proof $\pi \leftarrow \mathsf{NIZK}.\mathtt{Prove}(\mathsf{stmt}_{\mathsf{revoke}}; [r, \omega, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i])$ where $\mathsf{stmt}_{\mathsf{revoke}} = (i, \mathsf{gvk})$ and the relation is defined by: $\exists\ [r, \omega, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i]$: $\mathsf{com} = \mathsf{Com}(r; \omega)$ and $\mathsf{Table}_i = (\Gamma.\mathtt{Enc}((\mathsf{Com}(r_i; \omega_i), \mathsf{com}_{i,2}); \mathsf{ek}, \beta_i), \cdot, \mathtt{IN})$ and set $\mathsf{Table}_j = (\cdot, (i, \pi), \mathtt{OUT})$.
- Party i removes j from G and $(\{j\}, \mathsf{sign})$, $(\{j\}, \mathsf{revoke})$ from Π.

$\underline{\mathtt{Ver}(m, \sigma = (\mathsf{ct}, \pi); \mathsf{gvk} = (\mathsf{pp}, \mathsf{crs}, \mathsf{vk}, \mathsf{ek}, \mathsf{com}))}$:
Output $\mathsf{NIZK}.\mathtt{Verify}(\pi, \mathsf{stmt}_{\mathsf{sign}} = (m, \mathsf{ct}, \mathsf{gvk}), \mathsf{crs})$.

Figure 11: Our dynamic managerless group signature scheme $\sum_{\mathsf{Group}}^{\mathsf{w/o\ manager}}$ (Part 2/2).

signed using the private key of any authenticator in a group. The key advantages of the proposed method are: (i) the relying party server only needs to hold one public key per account which drastically reduces the verification costs compared to earlier solutions while still satisfying the unlinkability properties expected in FIDO, and (ii) a user can dynamically manage authenticators (e.g., add new or revoke existing authenticators from a group) on any client device without a strict dependency on the server. Given that the user interaction flow is similar to that of FIDO user authentication and the cryptographic methods do not impose significant delay, our study suggests that the proposed technique could be efficiently deployed in practice for authentication. Finally, we show a further construction of theoretical interest which achieves the further security notion of traceability not needed in the FIDO context.

References

[1] 1Password. How 1Password protects your data. https://support.1password.com/sync-options-security/, 2021. Online.

[2] FIDO Alliance. How FIDO Addresses a Full Range of Use Cases . https://media.fidoalliance.org/wp-content/uploads/2022/03/How-FIDO-Addresses-a-Full-Range-of-Use-Cases-March24.pdf, 2022. Online.

[3] Apple. Use Face ID on your iPhone or iPad Pro. https://support.apple.com/en-us/HT208109, 2020. [Online; accessed 25-May-2021].

[4] Apple. Use Touch ID on your iPhone or iPad Pro. https://support.apple.com/en-us/HT201371, 2020. [Online; accessed 25-May-2021].

[5] Apple. Accessing Keychain Items with Face ID or Touch ID. https://developer.apple.com/documentation/localauthentication/accessing_keychain_items_with_face_id_or_touch_id, 2021. Online.

[6] Apple. Supporting Passkeys. https://developer.apple.com/documentation/authenticationservices/public-private_key_authentication/supporting_passkeys, 2022. Online.

[7] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2000.

[8] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. Provable security analysis of FIDO2. Cryptology ePrint Archive, Report 2020/756, 2020. https://eprint.iacr.org/2020/756.

[9] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer, 2003.

[10] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2005.

[11] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.

[12] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick D. McDaniel, editors, *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*, pages 168–177. ACM, 2004.

[13] Bouncy castle, 2022. `https://www.bouncycastle.org/java.html`.

[14] Stéphane Ciolino, Simon Parkin, and Paul Dunphy. Of two minds about two-factor: Understanding everyday FIDO U2F usability through device comparison and experience sampling. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 339–356, 2019.

[15] Sanchari Das, Andrew Dingman, and L Jean Camp. Why johnny doesn't use two factor a two-phase usability study of the FIDO U2F security key. In *International Conference on Financial Cryptography and Data Security*, pages 160–179. Springer, 2018.

[16] Dashlane. Security at Dashlane. `https://support.dashlane.com/hc/en-us/articles/360012686840-FAQ-about-security-at-Dashlane`, 2021. Online.

[17] Emma Dauterman, Henry Corrigan-Gibbs, and David Mazières. SafetyPin: Encrypted backups with human-memorable secrets. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1121–1138, 2020.

[18] Haonan Feng, Hui Li, Xuesong Pan, and Ziming Zhao. A formal analysis of the FIDO UAF protocol. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

[19] FIDO Alliance. FIDO security reference, 2017. `https://fidoalliance.org/specs/fido-uafv1.1-ps-20170202/fido-security-ref-v1.1-ps-20170202.html`.

[20] FIDO Alliance. FIDO U2F overview. `https://fidoalliance.org/specs/u2f-specs-master/fido-u2f-overview.html`, 2021. [Online; accessed 25-June-2021].

[21] FIDO Alliance. FIDO UAF architectural overview. `https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-overview-v1.1-id-20170202.html`, 2021. [Online; accessed 25-June-2021].

[22] FIDO Alliance. Simpler, Stronger Authentication: Solving the World's Password Problem. `https://fidoalliance.org`, 2021. [Online; accessed 25-June-2021].

[23] Nick Frymann, Daniel Gardham, Franziskus Kiefer, Emil Lundberg, Mark Manulis, and Dain Nilsson. Asynchronous remote key generation: An analysis of yubico's proposal for w3c webauthn. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 939–954, 2020.

[24] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985.

[25] GitHub. Configuring two-factor authenticationxf recovery methods. `https://docs.github.com/en/github/authenticating-to-github/securing-your-account-with-two-factor-authentication-2fa/configuring-two-factor-authentication-recovery-methods`, 2021. Online.

[26] Oded Goldreich. *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press, 2001.

[27] Hidehito Gomi, Bill Leddy, and Dean H Saxe. Recommended account recovery practices for FIDO relying parties, February 2019.

[28] Google LLC. Google Authenticator. `https://tinyurl.com/rakk4ycy`, 2021. [Online; accessed 25-May-2021].

[29] HID Global. Advanced Multi-factor Authentication. `https://www.hidglobal.com/solutions/identity-access-management/advanced-multi-factor-authentication`, 2021. [Online; accessed 25-May-2021].

[30] Don Johnson, Alfred Menezes, and Scott A. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Sec.*, 2001.

[31] LastPass. LastPass Security History. `https://www.lastpass.com/security/what-if-lastpass-gets-hacked`, 2021. Online.

[32] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. Is FIDO2 the kingslayer of user authentication? a comparative usability study of FIDO2 passwordless authentication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 268–285. IEEE, 2020.

[33] Mark Manulis. Democratic group signatures: on an example of joint ventures. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 365–365, 2006.

[34] Microsoft. Microsoft Authenticator. `https://www.microsoft.com/en-us/account/authenticator`, 2021. [Online; accessed 25-May-2021].

[35] Deepak Mishra. ECDSA digital signature verification in java. `https://metamug.com/article/security/sign-verify-digital-signature-ecdsa-java.html`, 2019. Online.

[36] NIST Cybersecurity Insights. Out with the old, in with the new: making MFA the norm. `https://www.nist.gov/blogs/cybersecurity-insights/out-old-new-making-mfa-norm`, 2018. [Online; accessed 25-May-2021].

[37] Wataru Oogami and Max Hata. Multiple authenticators for reducing account recovery needs for FIDO-Enabled consumer accounts, June 2020.

[38] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptol.*, 13(3):361–396, 2000.

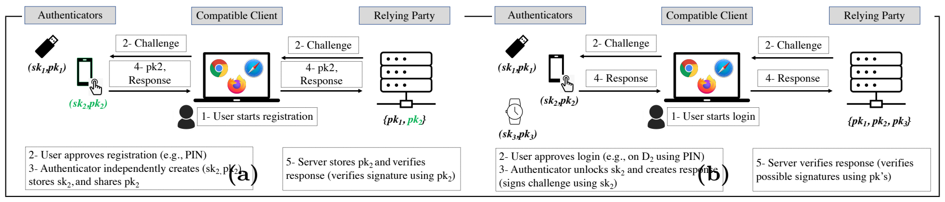
[39] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.

[40] Yannick Seurin. On the exact security of schnorr-type signatures in the random oracle model. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2012.

[41] Spongy castle, 2017. `https://rtyley.github.io/spongycastle/`.

[42] Microsoft Security Team. Building a world without passwords. `https://www.microsoft.com/security/blog/2018/05/01/building-a-world-without-passwords`, 2021. [Online; accessed 25-June-2021].

[43] Yubico. Have a Backup and Recovery Plan. `https://www.yubico.com/blog/backup-recovery-plan/`, 2021. Online.


Authenticators
Compatible Client
Relying Party
(sk1,pk1)
(sk2,pk2)
2- Challenge
4- pk2, Response
2- Challenge
4- pk2, Response
1- User starts registration
{pk1, pk2}
2- User approves registration (e.g., PIN)
3- Authenticator independently creates (sk2, pk2), stores sk2, and shares pk2
(a)
5- Server stores pk2 and verifies response (verifies signature using pk2)
Authenticators
Compatible Client
Relying Party
(sk1,pk1)
(sk2,pk2)
(sk3,pk3)
2- Challenge
4- Response
2- Challenge
4- Response
1- User starts login
{pk1, pk2, pk3}
2- User approves login (e.g., on D2 using PIN)
3- Authenticator unlocks sk2 and creates response (signs challenge using sk2)
(b)
5- Server verifies response (verifies possible signatures using pk's)
Figure 12: FIDO user authentication (a) registration, and (b) authentication steps.

A FIDO User Authentication

Figure 12 illustrates the registration and authentication steps during FIDO user authentication.

B FIDO Account Recovery Approaches

Table 3 provides a comparison of prevailing FIDO account recovery approaches.

C Group Signatures

This section describes the various security notions considered for group signatures in prior work, and also discusses group signatures in the dynamic setting (where the group changes with time).

Full-anonymity. Informally, anonymity requires that an adversary not in possession of the group manager's secret key should find it hard to recover the identity of the signer from a signature. Formally, this is defined as an indistinguishability requirement, on which is superimposed an adversary with strong attack capabilities. To capture the possibility of an adversary colluding with group members, the adversary is given the secret keys of all group members. To capture the possibility of the adversary seeing the results of previous openings by the group manager, they also have access to an opening oracle, $\mathtt{SigOpen}(\cdot, \cdot; \mathsf{gmsk})$, which when queried with a message m and signature σ, answers with $\mathtt{SigOpen}(m, \sigma; \mathsf{gmsk})$.

Full-traceability. Informally, traceability requires that signer anonymity can be revoked by the group manager. In particular, no colluding set X of group members (even consisting of the entire group, and even being in possession of the secret key for opening signatures) can create signatures that cannot be opened, or signatures that cannot be traced back to some member of the coalition. Note that giving the opening key to the adversary does not model corruption of the group manager, but rather compromise of the group manager's key by the adversary.

Dishonest group managers. The last consideration with regards to the definition pertains to the case where the group manager holding gmsk is not honest. If the group manager is truly dishonest, one should also assume that they do not behave as prescribed when applying the opening algorithm. For example, when asked to open a signature they can falsely accuse an arbitrary user, or claim that the signature cannot be opened. A solution to this problem is that when opening a signature, the group manager (on input opening key gmsk, message m and signature σ) outputs not only a user identity i, but also a "proof" π. This proof can be (publicly) verified by a "judging" algorithm $\mathtt{Judge}$ (which is added to the syntax of the group signature scheme,) such that if $\mathtt{SigOpen}(m, \sigma; \mathsf{gmsk}) = (i, \pi)$ then $\mathtt{Judge}(m, \sigma, i, \pi) = 1$. Within this framework, it is possible to formally capture security requirements regarding dishonest behavior of the group manager.

Relations to other security notions. Several security properties have been proposed in the context of group signatures thus far in the literature:

- *Unforgeability*: Only members of the group can create valid group signatures. Unforgeability is implied by full-traceability.

- *Anonymity*: Given a message and its signature, the identity of the individual signer cannot be determined without the group manager's secret key. Anonymity is implied by full-anonymity.

- *Traceability*: Given any valid signature, the group manager should be able to trace which user issued the signature. (This and the previous requirement imply that only the group manager can break users' anonymity.) Traceability is implied by full-traceability.

- *Unlinkability*: Given two messages and their signatures, we cannot tell if the signatures were from the same signer or not. Anonymity and unlinkability are essentially the same property.

- *No framing*: Even if all other group members (and the manager) collude, they cannot forge a signature for a non-participating group member. This property is also referred to *exculpability.* No framing is implied by full-traceability.

- *Coalition resistance*: A colluding subset of group members cannot generate a valid signature that the group manager cannot link to one of the colluding group members. Coalition resistance is implied by full-traceability.

Dynamic Group Signatures. We next consider the setting of dynamic group signature schemes with a group manager. In an incremental group signature scheme (groups supporting an add operation), the key generation algorithm produces (beside the group public key gvk) two secret keys: an *issuing key* gmik and an opening key gmok. No signing keys gsk are output by the key generation algorithm. The two keys gmik, gmok are given to two different group managers, one of which has authority on the group membership, and the other has authority on traceability. Using gmik, the key issuer can generate (possibly

Table 3: Comparison of prominent methods for FIDO account recovery.

Recovery Approach	Method	Summary	Advantages	Disadvantages
Trust Re-establishment	One-Time Passcode	Two factor authentication using one-time PIN/passcode over a backup channel, e.g., email or SMS.	- Users are habituated - No additional burden on users to manage a backup code/password	- Susceptible to phishing
	Backup codes [25]	Generated at the time of account creation/recovery; setup and stored physically or digitally by user.	- For physical storage, user doesn't need to trust a credential manager or account management service for securing their backup credentials	- Prone to being forgotten, lost or stolen - Susceptible to phishing
	ID document verification, FIDO recommended [27]	Identity proofing using pictures of a user's identity document and selfie.	- ID documents are a stronger root of trust compared to OTP/backup codes - Convenient user experience	- Unreliable in different capture settings, e.g., illumination - Susceptible to spoofing, e.g., digital deepfakes
	Multi-device credentials, FIDO recommended [2]	Backup and sync FIDO credentials (secret keys) across multiple devices using platform provider's (e.g., Apple, Google) cloud.	- Users are habituated to storing other credentials (e.g., passwords, credit card details) on platform providers cloud - Convenient user experience (set and forget, automatic sync, backup)	- Platform provider can track FIDO credential usage across different relying parties - FIDO credentials not truly decentralized; their security defaults to platform provider's cloud.
Enrolling additional authenticators	Yubico [43], FIDO recommended [27, 37]	Relying party holds public key for all registered authenticators and verifies responses against all public keys.	- Not prone to phishing	- Requires user to register additional authenticators with each relying party - Management of multiple keys by relying party server
	Yubico (proposal) [23]	Backup authenticator performs asynchronous key agreement with primary authenticator	- Not prone to phishing	- Requires user to register additional authenticators with each relying party - Requires additional communication with relying party servers
	Our Method	Authenticator Groups with each authenticator having unique private key, and relying party server having a group verification key	- Relying party only needs one verification key per account - Addition and removal of additional authenticators without interaction with relying party servers - Same authenticator group can be registered and used across multiple relying parties	- Requires user to create and manage authenticator groups

via an interactive process) signing keys gsk_i and distribute them to prospective group members. In terms of security definitions, the key gmik is given to the adversary in the definition of anonymity, but not in the definition of traceability, as knowledge of this key would allow to generate "dummy" group members and use their keys to sign untraceable messages. Alternatively, one can postulate that if the signature opener cannot trace a signature, then he will blame the key issuer. Although the above definition allows the group to change over time, the security properties are still static: a signer i that joins the group at time t, can use the newly acquired key gsk_i to sign documents that predate time t. This problem can be easily solved enhancing the signatures with an explicit time counter, and using the technique of forward security. Forward security for group signatures is defined using a key evolution paradigm. The lifetime of the public key is divided into time periods, and signing keys of the group members change over time, with the key of user i at time t denoted by $\mathsf{gsk}_i[t]$. At the end of each time period, each user updates his key using an update algorithm $\mathsf{gsk}_i[t+1] = \mathsf{Upd}(\mathsf{gsk}_i[t])$. A forward secure group signature schemes requires that an attacker should not be able to produce valid signatures for any earlier time period. A (forward) secure incremental group signature scheme supporting the add operation is obtained letting the key issuer generate key $\mathsf{gsk}_i[t]$ when user i joins the group at time t.

Finally, we consider *fully dynamic* group signature schemes, that are both incremental and decremental (groups supporting a revoke operation). Here, a question that immediately comes up is when should a signature generated by a revoked member be accepted by the signature verification algorithm? There are two possible answers: (1) if they were a group member when then signature was generated, or (2) if they belonged to the group at the time verification algorithm is invoked.[11] Clearly, there is no "right" answer, and what definition should be used depends on the application. In either case, different kinds of inefficiency are necessarily introduced in the protocol. More formally, consider a signature σ produced (using key gsk_i) by some user i who belonged to the group at time t_1, but not at times t_0 and t_2, Now, say σ is verified at time $t \in \{t_0, t_1, t_2\}$ (using key $\mathsf{gvk}[t]$, where gvk denotes the verification key at time t). In case (2), σ should be accepted using $\mathsf{gvk}[t_1]$, but not using $\mathsf{gvk}[t_0]$ or $\mathsf{gvk}[t_2]$. In particular, the public keys $\mathsf{gvk}[t]$ must be different. This is undesirable because it requires the verifier to continuously communicate with the group manager to update the group public key. Moreover, this definition raises potential anonymity problems: by verifying the same signature against different public keys $\mathsf{gvk}[t]$, one can determine when the signer joined and left the group, and possibly use this information to discover the signer identity. Consider case (1), where signatures are valid only if signer belongs to the group at the time the signature is issued. Now, σ should be accepted using $\mathsf{gvk}[t_0]$, $\mathsf{gvk}[t_1]$ and $\mathsf{gvk}[t_2]$. This time, the public key may stay the same throughout the lifetime of the group, but in order to achieve forward security, the update function Upd should not be publicly

[11]Notice that in the first case, signatures should remain valid (and the signer anonymous) even after the signer leaves the group, while in the second case removing the signer from the group should immediately invalidate all of its signatures.

computable by the group members. This introduces inefficiency, as the group members now need to interact with the key issuer to update their signing key from one time period to the next.

D Cryptographic Preliminaries

Throughout the paper, we use λ to denote the security parameter. An algorithm $\mathcal{T}$ is said to be PPT if it is modeled as a probabilistic Turing machine that runs in time polynomial in λ. Informally, we say that a function is negligible if it vanishes faster than the inverse of any polynomial. If S is a set, then $x \leftarrow S$ indicates the process of selecting x uniformly at random over S (which in particular assumes that S can be sampled efficiently). Similarly, $x \leftarrow \mathcal{T}(\cdot)$ denotes the random variable that is the output of a randomized algorithm $\mathcal{T}$. Furthermore, $x = \mathcal{T}(\cdot; r)$ denotes the output of a randomized algorithm $\mathcal{T}$ with randomness r.

We use the following cryptographic primitives in our constructions. We refer to [26] for their formal security properties.

Commitment. A commitment scheme consists of the following algorithms:

- $\texttt{Gen}(1^\lambda)$: Samples public parameters pp.
- $\mathsf{Com}(m; \omega)$: Generates a commitment com to m using randomness ω.
- $\mathsf{Open}(\mathsf{com}; \omega)$: Opens the commitment com using the opening randomness ω to obtain m.

Public-Key Encryption. A public-key encryption scheme consists of the following algorithms:

- $\texttt{KeyGen}(1^\lambda)$: Samples a public key ek and secret key dk.
- $\texttt{Enc}(m; \mathsf{ek})$: Generates a ciphertext ct by encrypting the plaintext m.
- $\texttt{Dec}(\mathsf{ct}; \mathsf{dk})$: Decrypts ciphertext ct to obtain the plaintext m.

Non-interactive Zero-Knowledge Argument (NIZK). A NIZK consists of the following algorithms:

- $\texttt{CRSGen}(1^\lambda)$: Generates a common reference string crs.
- $\texttt{Prove}(\mathsf{stmt}, \mathsf{crs}; \mathsf{w})$: Generates a proof π of the statement stmt using crs and witness w.
- $\texttt{Verify}(\pi, \mathsf{stmt}, \mathsf{crs})$: Verifies proof π using crs and outputs 0 or 1.

In our use of a NIZK, we suppress crs for ease of exposition.

E Signing Key Re-randomizable Signatures

E.1 Re-randomizable Schnorr Signatures

Recall the modified Schnorr signature scheme.

- $\texttt{KeyGen}(1^\lambda)$: Samples a random $x \leftarrow \mathbb{Z}_{Mq}$ and sets $\mathsf{sk} = x$ and $\mathsf{vk} = g^{x \mod q}$.
- $\texttt{Sign}(m, \mathsf{sk} = x)$: Samples a random $r \leftarrow \mathbb{Z}_q$ and sets $e = \texttt{Hash}(g^r \| m)$, $s = (r - xe) \mod q$ and $\sigma = (s, e)$.
- $\texttt{Ver}(m, (s, e), \mathsf{vk})$: Checks if $e = \texttt{Hash}(g^s \mathsf{vk}^e \| m)$.

We now prove that the above modified signature scheme is existentially unforgeable. Recall that a signature scheme $\sum$ is said to be *existentially unforgeable* if, for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$, the advantage $\mathbf{Adv}^{\textbf{Exist-Forge}}_{\sum,\mathcal{A},\lambda}$ of $\mathcal{A}$, defined by the probability term below, is negligible in λ:

$$\Pr\left[\begin{array}{c} (\mathsf{vk}, \mathsf{sk}) \leftarrow \sum.\texttt{KeyGen}(1^\lambda) \\ \mathcal{Q} = \emptyset \\ \left\{\begin{array}{c} m_i \leftarrow \mathcal{A}(1^\lambda, \mathcal{Q}) \\ \sigma_i \leftarrow \sum.\texttt{Sign}(m_i; \mathsf{sk}) \\ \mathcal{Q} = \mathcal{Q} \cup \{(m_i, \sigma_i)\} \end{array}\right\}_i \\ (m, \sigma) \leftarrow \mathcal{A}(1^\lambda, \mathcal{Q}) \end{array} : \begin{array}{c} \sum.\texttt{Ver}(m, \sigma; \mathsf{vk}) = 1 \\ \wedge (m, \sigma) \notin \mathcal{Q} \end{array}\right]$$

Suppose $\sum^*_{\mathsf{Schnorr}}$ is *existentially forgeable*. That is, suppose there exists an adversary $\mathcal{A}$ such that $\mathbf{Adv}^{\textbf{Exist-Forge}}_{\sum^*_{\mathsf{Schnorr}},\mathcal{A},\lambda}$ is non-negligible in λ. We construct an adversary $\mathcal{B}$ such that $\mathbf{Adv}^{\textbf{Exist-Forge}}_{\sum_{\mathsf{Schnorr}},\mathcal{B},\lambda}$ is also non-negligible in λ. Since we know that no such $\mathcal{B}$ exists, this proves that $\sum^*_{\mathsf{Schnorr}}$ is existentially unforgeable. Let $\mathcal{C}$ be the challenger of the existential forgeability game for $\sum_{\mathsf{Schnorr}}$. $\mathcal{C}$ begins by running $(\mathsf{vk}, \mathsf{sk}) \leftarrow \sum_{\mathsf{Schnorr}}.\texttt{KeyGen}(1^\lambda)$ and sends vk to $\mathcal{B}$. $\mathcal{B}$ then forwards vk to $\mathcal{A}$. Note that this first message is identically distributed with respect to the vk that $\mathcal{A}$ must receive for the existential forgeability game of $\sum^*_{\mathsf{Schnorr}}$. After this, for every message m_i that $\mathcal{A}$ issues a signature query for, $\mathcal{B}$ forwards m_i to $\mathcal{C}$ and obtains a signature σ_i which $\mathcal{B}$ forwards to $\mathcal{A}$. We now proceed to argue that signatures for $\sum_{\mathsf{Schnorr}}$ produced by $\mathcal{C}$ using signing key $\mathsf{sk} = x \in \mathbb{Z}_q$ are valid signatures for $\sum^*_{\mathsf{Schnorr}}$ as well. Consider any $x^* \in \mathbb{Z}_{Mq}$ such that $x^* = x \mod q$. We show that $\mathcal{B}$'s interaction with $\mathcal{A}$ is consistent with a world where $\mathcal{B}$ is working with signing key $\mathsf{sk}^* = x^*$. First, note that the verification key corresponding to sk^* is $\mathsf{vk}^* = g^{x^*} = g^{x^* \mod q} = g^x$ since g is a generator of the group $\mathbb{G}$ of order q. Next, let $\sigma_i \leftarrow \sum_{\mathsf{Schnorr}}.\texttt{Sign}(m_i; \mathsf{sk} = x)$. That is, $\sigma_i = (s_i, e_i)$, where $e_i = \texttt{Hash}(g^{r_i} \| m_i)$ for some random $r_i \leftarrow \mathbb{Z}_q$ sampled by $\mathcal{C}$, and $s_i = r_i - xe_i$. Now, let $\sigma^*_i \leftarrow \sum^*_{\mathsf{Schnorr}}.\texttt{Sign}(m_i; \mathsf{sk}^* = x^*)$. That is, let $\sigma^*_i = (s^*_i, e^*_i)$, where $e^*_i = \texttt{Hash}(g^{r^*_i} \| m_i)$ for some random $r^*_i \leftarrow \mathbb{Z}_q$ which was to be sampled by $\mathcal{B}$, and $s^*_i = r^*_i - x^* e^*_i$. Let us suppose that $r^*_i = r_i$. Then, we have $e^*_i = e_i$ and $s^*_i = (r_i - x^* e_i) \mod q = r_i - xe_i = s_i$. That is, $\sigma^*_i = \sigma_i$ is a valid signature for $\sum^*_{\mathsf{Schnorr}}$. Finally, we know that $\mathcal{A}$ outputs a pair (m, σ)

such that $(m,\sigma) \neq (m_i,\sigma_i)$ for any i, and $\sum^*_{\mathsf{Schnorr}}.\mathtt{Ver}(m,\sigma;\mathsf{vk}) = 1$. This implies that $\sum_{\mathsf{Schnorr}}.\mathtt{Ver}(m,\sigma;\mathsf{vk}) = 1$. Thus, $\mathcal{B}$ simply forwards (m,σ) to $\mathcal{C}$. We have that $\mathbf{Adv}^{\textbf{Exist-Forge}}_{\sum_{\mathsf{Schnorr}},\mathcal{B},\lambda} = \mathbf{Adv}^{\textbf{Exist-Forge}}_{\sum^*_{\mathsf{Schnorr}},\mathcal{A},\lambda}$. This completes our reduction and shows that $\sum^*_{\mathsf{Schnorr}}$ is existentially unforgeable.

E.2 Re-randomizable El Gamal Signatures

Let $\sum_{\mathsf{ElGamal}}$ be the **El Gamal signature scheme** [24] over the multiplicative group $\mathbb{Z}^*_q$ modulo prime $q = \mathcal{O}(2^\lambda)$ with generator g, consisting of the following algorithms:

- $\mathtt{KeyGen}(1^\lambda)$: Samples a random $x \leftarrow \mathbb{Z}^*_q$ and sets $\mathsf{sk} = x$ and $\mathsf{vk} = g^x$.
- $\mathtt{Sign}(m;\mathsf{sk} = x)$: Samples a random $r \leftarrow \mathbb{Z}^*_q$ such that $(r, q-1) = 1$ so that $r^{-1} \mod (q-1)$ exists. Set $s = g^r$, $e = r^{-1}(\mathtt{Hash}(m) - xs) \mod (q-1)$ and $\sigma = (s,e)$.
- $\mathtt{Ver}(m,(s,e);\mathsf{vk})$: Checks if $g^{\mathtt{Hash}(m)} = \mathsf{vk}^s s^e$.

$\sum_{\mathsf{ElGamal}}$ is secure in the random oracle model based on the discrete logarithm assumption [38].

The El Gamal signature scheme can be modified in a similar manner. Let $\sum^*_{\mathsf{ElGamal}}$ be the **modified El Gamal signature scheme** over the multiplicative group $\mathbb{Z}^*_q$ modulo the prime $q = \mathcal{O}(2^\lambda)$ with generator g, consisting of the following algorithms:

- $\mathtt{KeyGen}(1^\lambda)$: Samples a random $x \leftarrow \mathbb{Z}_{M(q-1)}$ and sets $\mathsf{sk} = x$ and $\mathsf{vk} = g^{x \mod (q-1)}$.
- $\mathtt{Sign}(m;\mathsf{sk} = x)$: Samples a random $r \leftarrow \mathbb{Z}^*_q$ such that $(r, q-1) = 1$ so that $r^{-1} \mod (q-1)$ exists and sets $s = g^r$, $e = r^{-1}(\mathtt{Hash}(m) - xs) \mod (q-1)$ and $\sigma = (s,e)$.
- $\mathtt{Ver}(m,(s,e);\mathsf{vk})$: Checks if $g^{\mathtt{Hash}(m)} = \mathsf{vk}^s s^e$.

Figure 13 describes a protocol Π_{ElGamal} that securely realizes $\mathcal{F}_{\mathbf{ReRand}}$ for the above scheme. As in the previous case, the security of Π_{ElGamal} is easy to observe from the correctness of the modified El Gamal signature scheme. We also prove that the modified signature scheme is existentially unforgeable.

F Proof of Security

Theorem 1. *Assuming* Com *is a computationally hiding and perfectly binding commitment scheme,* $\sum_{\mathsf{SR}}$ *is a secure signing key re-randomizable signature scheme,* Γ *is a CCA2-secure public key encryption scheme and* NIZK *is a a non-interactive zero-knowledge argument, scheme* $\sum^{\mathsf{w/o\ manager}}_{\mathsf{Group}}$ *(Figure 11) is a dynamic managerless group signature scheme that satisfies Definition 1 with respect to group structure* Π_{uni}.

Π_{ElGamal}

A has a key-pair $(\mathsf{vk}, \mathsf{sk_A}) \leftarrow \sum^{*}_{\mathsf{ElGamal}}.\mathtt{KeyGen}(1^{\lambda})$
B wants a random signing key $\mathsf{sk_B}$ corresponding to vk.

Inputs: $\mathsf{pub} = (\mathbb{G}, q, M, \mathsf{vk})$, $\mathsf{inp_A} = \mathsf{sk_A}$, $\mathsf{inp_B} = \perp$

Protocol:

1. A computes and sends $x = \mathsf{sk_A} \mod (q-1)$ to B.
2. B samples random $y \leftarrow \mathbb{Z}_M$.
3. B outputs $\mathsf{sk_B} = x + y(q-1)$.

Outputs: $\mathsf{outp_A} = \perp$, $\mathsf{outp_B} = \mathsf{sk_B}$

Figure 13: Protocol Π_{ElGamal} for realizing $\mathcal{F}_{\mathbf{ReRand}}$ in the modified El Gamal signature scheme.

Proof. We now prove that our scheme satisfies each of the four properties. As in the definition, for brevity, we denote $\sum = \sum^{\mathsf{w/o\ manager}}_{\mathsf{Group}}$.

Unforgeability. Recall that an adversary $\mathcal{A}$ breaks the unforgeability property if it outputs (m^*, σ^*, X^*) such that, with non-negligible probability: $\sum.\mathtt{Ver}(m^*, \sigma^*; \mathsf{gvk}) = 1$, $Z \cap H \neq \emptyset$ where $Z \leftarrow \sum.\mathtt{SigOpen}(m^*, \sigma^*; \mathsf{ok}_{X^*})$ and (m^*, σ^*) was not previously output by the signature oracle. We prove security using a hybrid argument.

- Hyb_0: This corresponds to the original experiment.
- Hyb_1: In this hybrid, challenger $\mathcal{C}$ does the following: for any invocation of protocol Π_{ReRand} as part of protocol $\mathtt{AddMember}$ where a corrupt party j is trying to add an honest user i, instead of running the honest execution of Π_{ReRand}, $\mathcal{C}$ runs the simulator $\mathsf{Sim}_{\mathbf{ReRand}}$ on behalf of the user i.

  Since protocol Π_{ReRand} securely realizes functionality $\mathcal{F}_{\mathbf{ReRand}}$ for signature scheme $\sum_{\mathsf{SR}}$, it is easy to observe that for any adversary $\mathcal{A}$, its advantage in outputting a tuple (m^*, σ^*, X^*) that breaks the unforgeability game remains negligibly close to its advantage in the original experiment Hyb_0.
- Hyb_2: In this hybrid, on behalf of any honest party i, whenever running the signing algorithm $\mathtt{Sign}$, challenger $\mathcal{C}$ runs the simulator of the NIZK to generate the proof π. $\mathcal{C}$ also generates a simulated CRS in the key generation step.

  From the zero knowledge property of the NIZK, for any adversary $\mathcal{A}$, its advantage in outputting a tuple (m^*, σ^*, X^*) that breaks the unforgeability game remains negligibly close to its advantage in Hyb_1.

We now argue that in Hyb_2, $\mathcal{A}$ does not break the unforgeability property and this completes the proof. Let's assume for the sake of contradiction that there exists an adversary $\mathcal{A}$ that manages to break the unforgeability probability in Hyb_2. First, since the signature σ^* verifies successfully, from the soundness of the NIZK argument, with overwhelming probability, $\exists\ (i, \sigma', r, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i)$, such that:

- $\sum_{\mathsf{SR}}.\mathtt{Ver}(m^*, \sigma'; \mathsf{vk}) = 1$
- $\mathsf{ct} = \Gamma.\mathtt{Enc}([i, \sigma']; \mathsf{ek}, r)$
- $\mathsf{Table}_i = (\Gamma.\mathtt{Enc}((\mathsf{Com}(r_i; \omega_i), \mathsf{com}_{i,2}); \mathsf{ek}, \beta_i), \cdot, \mathtt{IN})$

From the correctness of the decryption algorithm $\Gamma.\mathtt{Dec}$ and the verification algorithm $\mathsf{NIZK}.\mathtt{Verify}$, $\sum.\mathtt{SigOpen}$ outputs this index i with overwhelming probability. Since $\mathcal{A}$ wins the game, it must be the case that $(m^*, \sigma^*, \cdot) \notin \mathsf{List}$ and $i \cap H \neq \emptyset$. From the correctness of the encryption scheme Γ, except with negligible probability, there doesn't exist another tuple $(\mathsf{Com}(r_j; \omega_j), \mathsf{com}_{j,2}, \beta_j)$ with $j \neq i$ such that $\mathsf{Table}_i = \Gamma.\mathtt{Enc}((\mathsf{Com}(r_j; \omega_j), \mathsf{com}_{j,2}); \mathsf{ek}, \beta_j)$. We now use this adversary to build a reduction $\mathcal{B}$ that breaks the hiding of the commitment scheme. $\mathcal{B}$ interacts with a challenger $\mathcal{C}_{\mathsf{com}}$ of the commitment scheme and for every honest party i, it sends a pair $(\mathsf{rand}_i, 0)$ to $\mathcal{C}_{\mathsf{com}}$. $\mathcal{C}_{\mathsf{com}}$ tosses a bit b and if $b = 0$, responds back with a commitment to rand_i for each i and if $b = 1$, responds with a commitment to 0 for each i. For each i, $\mathcal{B}$ sets this commitment received as the value $\mathsf{com}_{i,1}$ and interacts with the adversary $\mathcal{A}$ exactly as done by challenger $\mathcal{C}$. Now, since $\mathcal{A}$ successfully breaks the unforgeability property, $\mathcal{B}$ runs the extractor Ext of the NIZK on the proof π^* that $\mathcal{A}$ outputs as part of σ^* to recover witness $(i, \sigma', r, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i)$. Thus, $\mathcal{B}$ learns whether $r_i = \mathsf{rand}_i$ or 0 and can break the hiding of the commitment scheme with overwhelming probability which is a contradiction. This completes the proof.

Anonymity. In the anonymity game, one of the conditions for any $\mathcal{A}$ to win is that the following is not true: $\exists\ X \subseteq (G \cup R) \setminus H : (X, \mathsf{open}) \in \Pi$. For the group structure Π_{uni}, this means that for every corrupt party j, $(j, \mathsf{open}) \notin \Pi$: $\mathcal{A}$ does not learn the opening key for any party. This essentially translates to the fact that the adversary does not corrupt any party in the system because any party once added, never loses opening privileges (even on being revoked). Having noted this observation, we now prove the anonymity property via a hybrid argument.

- Hyb_0: This corresponds to the original anonymity experiment where the challenger $\mathcal{C}$ picks bit $b = 0$.
- Hyb_1: In this hybrid, on behalf of any honest party i, whenever generating a NIZK (as part of $\sum.\mathtt{AddMember}, \sum.\mathtt{Sign}$ or $\sum.\mathtt{RevokeMember}$), challenger $\mathcal{C}$ runs the simulator of the NIZK to generate the proof. $\mathcal{C}$ also generates a simulated CRS in the key generation step.

From the zero knowledge property of the NIZK, Hyb_1 is computationally indistinguishable from Hyb_0.

- Hyb_2: For any invocation of protocol Π_{ReRand} as part of protocol $\sum.\mathtt{AddMember}$ where an honest party i is trying to add a corrupt j, instead of running the honest execution of Π_{ReRand}, $\mathcal{C}$ runs the simulator $\mathsf{Sim}_{\mathbf{ReRand}}$ on behalf of the user i.

  Since protocol Π_{ReRand} securely realizes functionality $\mathcal{F}_{\mathbf{ReRand}}$ for signature scheme $\sum_{\mathsf{SR}}$, Hyb_2 is computationally indistinguishable from Hyb_1.

- Hyb_3: In protocol $\sum.\mathtt{KeyGen}$ and on behalf of every honest party in protocol $\sum.\mathtt{AddMember}$, $\mathcal{C}$ generates every commitment as a commitment to 0 instead of as in the original protocol.

  From the hiding property of the commitment scheme, Hyb_3 is computationally indistinguishable from Hyb_2.

- Hyb_4: In this hybrid, on behalf of every honest party i, $\mathcal{C}$ does not use the tuple $(r, \omega, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i)$ anymore. That is, the revoking key rk_i is not used as part of algorithm $\sum.\mathtt{RevokeMember}$ and except for sk_i, the rest of ik_i and gsk_i are not used for $\sum.\mathtt{AddMember}$ and $\sum.\mathtt{Sign}$ respectively.

  This hybrid is just a syntactic change. Observe that since the NIZK proofs and protocol Π_{ReRand} were anyway already simulated, in the previous hybrid too, the values $(r, \omega, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i)$ were not being used. Thus, Hyb_4 is identical to Hyb_3.

- Hyb_5: In this hybrid, as part of computing the challenge signature $\mathsf{Chall} = (m^*, \sigma^*)$, the term σ' is computed as $\sigma' = \sum_{\mathsf{SR}}.\mathtt{Sign}(m^*; \mathsf{sk}_1)$ instead of $\sigma' = \sum_{\mathsf{SR}}.\mathtt{Sign}(m^*; \mathsf{sk}_0)$.

  As observed in the proof of the warm-up construction in Section 4.1, for the signing key re-randomizable signature scheme $\sum_{\mathsf{SR}}$, for any m^* chosen by the adversary, the distributions $\{\sum_{\mathsf{SR}}.\mathtt{Sign}(m^*; \mathsf{sk}_0)\}$ are $\{\sum_{\mathsf{SR}}.\mathtt{Sign}(m^*; \mathsf{sk}_1)\}$ are identical. Thus, Hyb_5 is identical to Hyb_4.

- Hyb_6: In this hybrid, as part of computing the challenge signature $\mathsf{Chall} = (m^*, \sigma^*)$, the term ct is computed as $\mathsf{ct} = \Gamma.\mathtt{Enc}([1, \sigma']; \mathsf{ek}, r)$ instead of as an encryption of $(0, \sigma')$.

  The only difference between the two hybrids is in the generation of the ciphertext ct that is part of σ^*. If there exists an adversary $\mathcal{A}$ that can distinguish between these two hybrids with non-negligible probability, we build a reduction $\mathcal{B}$ that can break the CCA security of the encryption scheme Γ. $\mathcal{B}$ interacts with a challenger $\mathcal{C}_\Gamma$ as part of the CCA game. Upon receiving public key ek, it begins an interaction with $\mathcal{A}$ as in Hyb_5. $\mathcal{B}$ runs the $\mathtt{KeyGen}$ as in Hyb_5 except that it sets ek using the value received from $\mathcal{C}_\Gamma$ and does not set any dk. Observe that $\mathcal{A}$ does not learn dk at all since it is not allowed to corrupt any party to win the anonymity game. Now, whenever $\mathcal{B}$ wishes to run $\Gamma.\mathtt{Dec}$ (as part of $\sum.\mathtt{SigOpen}$), it uses the decryption oracle provided by $\mathcal{C}_{\mathtt{Enc}}$ to do so. $\mathcal{B}$ submits tuple $(0, \sigma')$ and $(1, \sigma')$ to $\mathcal{C}_\Gamma$ where σ' is generated as the first part of the challenge signature $\mathsf{Chall} = (m^*, \sigma^*)$. It sets the ciphertext ct^* received from $\mathcal{C}_\Gamma$ as the ciphertext ct in the challenge signature σ^* and proceeds interacting with $\mathcal{A}$. If $\mathcal{A}$ guesses Hyb_5, then $\mathcal{B}$ guesses that ct^* is an encryption of $(0, \sigma')$ and if $\mathcal{A}$ guesses Hyb_6, $\mathcal{B}$ guesses that ct^* is an encryption of $(1, \sigma')$. Thus, observe that if there exists an adversary $\mathcal{A}$ that can distinguish between these two hybrids with non-negligible probability, there exists a reduction that breaks the CCA security of the encryption scheme Γ which is a contradiction.

- Hyb_7: In protocol $\sum.\mathtt{KeyGen}$ and on behalf of every honest party in protocol $\mathtt{AddMember}$, $\mathcal{C}$ generates every commitment honestly as in the original protocol instead of as a commitment to 0. Observe that implicitly, part of the tuple $(r, \omega, r_i, \omega_i, \mathsf{com}_{i,2}, \beta_i)$ is also used now.

  As before, from the hiding property of the commitment scheme, Hyb_7 is computationally indistinguishable from Hyb_6.

- Hyb_8: For any invocation of protocol Π_{ReRand} as part of protocol $\sum.\mathtt{AddMember}$ where an honest party i is trying to add a corrupt j, $\mathcal{C}$ runs an honest execution of Π_{ReRand} instead of running the simulator $\mathsf{Sim}_{\mathbf{ReRand}}$.

  As before, since protocol Π_{ReRand} securely realizes functionality $\mathcal{F}_{\mathbf{ReRand}}$ for signature scheme $\sum_{\mathsf{SR}}$, Hyb_8 is computationally indistinguishable from Hyb_7.

- Hyb_9: In this hybrid, on behalf of any honest party i, whenever generating a NIZK (as part of $\sum.\mathtt{AddMember}$, $\sum.\mathtt{Sign}$ or $\sum.\mathtt{RevokeMember}$), challenger $\mathcal{C}$ runs the honest prover algorithm $\mathsf{NIZK}.\mathtt{Prove}$ to generate the proof instead of the simulator. $\mathcal{C}$ also generates an honestly generated CRS in the key generation step. This corresponds to the original anonymity experiment where the challenger $\mathcal{C}$ picks bit $b = 1$.

  As before, from the zero knowledge property of the NIZK, Hyb_9 is computationally indistinguishable from Hyb_8.

Finally, observe that since Hyb_0 is computationally indistinguishable from Hyb_9, the adversary's advantage in guessing bit b' in the anonymity game is negligibly close to 1/2 and this completes the proof.

Revocability. Suppose, for the sake of contradiction, there exists an adversary $\mathcal{A}$ that breaks the revocability property. That is, it outputs (m^*, σ^*, X^*)

such that, with non-negligible probability: $\sum.\mathtt{Ver}(m^*, \sigma^*; \mathsf{gvk}) = 1$, $i^* \notin G$ where $i^* \leftarrow \sum.\mathtt{SigOpen}(m^*, \sigma^*; \mathsf{ok}_{X^*})$. From the correctness of the decryption algorithm $\Gamma.\mathtt{Dec}$ and the verification algorithm $\mathsf{NIZK}.\mathtt{Verify}$, observe that i^* is indeed same as the identity of the party who generated σ^*. That is, $\sigma^* = \sum.\mathtt{Sign}(m, \mathsf{gsk}_{i^*})$. Since $\sum.\mathtt{Ver}(m^*, \sigma^* = (\mathsf{ct}^*, \pi^*); \mathsf{gvk}) = 1$, $\mathsf{NIZK}.\mathtt{Verify}(\pi^*, \mathsf{stmt}_{\mathsf{sign}} = (m^*, \mathsf{ct}^*, \mathsf{gvk}), \mathsf{crs}) = 1$. This means that Table_{i^*} is of the form $(\cdot, \cdot, \mathtt{IN})$. However, observe that if a party $i \notin G$, either i has never been added to G or it has been added and subsequently revoked. In other words, $\mathsf{Table}_{i^*} = \perp$ or $\mathsf{Table}_{i^*} = (\cdot, \cdot, \mathtt{OUT})$. Thus, if there exists an adversary $\mathcal{A}$ that breaks the revocability property, we can use $\mathcal{A}$ to build a reduction $\mathcal{B}$ that breaks the soundness of the NIZK argument with non-negligible probability which is a contradiction. This completes the proof.

Traceability. To prove traceability, we argue that for any adversary $\mathcal{A}$, each of the winning conditions occurs only with negligible probability. Let $(m^*, \sigma^*, X^*, Y^*, r)$ be the tuple output by the adversary. Let $Z' \leftarrow \sum.\mathtt{SigOpen}(m^*, \sigma^*; \mathsf{ok}_{X^*})$, $Z'' \leftarrow \sum.\mathtt{SigOpen}(m^*, \sigma^*; \mathsf{ok}_{Y^*})$ and $Z''' \leftarrow \sum.\mathtt{SigOpen}(m^*, \sum.\mathtt{Sign}(m^*; \mathsf{sk}_{X^*}, r); \mathsf{ok}_{Y^*})$.

1. $\{(m^*, \sigma^*, X^*), (m^*, \sigma^*, Y^*)\} \subset \mathsf{List}$:
   Suppose $(m^*, \sigma^*, X^*) \in \mathcal{L}$. From the definition of algorithm $\sum.\mathtt{Sign}$, $\sigma^* = (\mathsf{ct}^*, \pi^*)$ where $\mathsf{ct}^* = \Gamma.\mathtt{Enc}([X^*, \sigma']; \mathsf{ek}, r^*)$ using some randomness r^* and $\sigma' = \sum_{\mathsf{SR}}.\mathtt{Sign}(m; \mathsf{sk}_{X^*})$. From the correctness of encryption scheme Γ, $\mathsf{ct}^* \neq \Gamma.\mathtt{Enc}([Y^*, \cdot]; \mathsf{ek}, \cdot)$. Thus, $(m^*, \sigma^*, Y^*) \not\subset \mathcal{L}$.

2. $Z' = \perp$ **or** $Z'' = \perp$ **or** $Z''' = \perp$:
   Since $\sum.\mathtt{Ver}(m^*, \sigma^*; \mathsf{gvk}) = 1$, from the soundness of the NIZK argument, $\sigma^* = (\mathsf{ct}^*, \pi^*)$ where $\mathsf{ct}^* = \Gamma.\mathtt{Enc}((\cdot, \cdot); \mathsf{ek}, \cdot)$. Thus, both $Z' \neq \perp$ and $Z'' \neq \perp$. Similarly, since $\sum.\mathtt{Sign}(m^*; \mathsf{sk}_{X^*}, r)$ is generated honestly, $Z''' \neq \perp$.

3. $Z' \neq Z''$:
   From the description of our scheme $\sum$, $\mathsf{ok}_{X^*} = \mathsf{ok}_{X^*} = \mathsf{dk}$. Thus, $\sum.\mathtt{SigOpen}(m^*, \sigma^*; \mathsf{ok}_{X^*}) = \sum.\mathtt{SigOpen}(m^*, \sigma^*; \mathsf{ok}_{Y^*})$ and hence $Z' = Z''$.

4. $(m^*, \sigma^*, \cdot) \in \mathsf{List}$ and $(m^*, \sigma^*, Z') \notin \mathsf{List}$:
   Suppose $(m^*, \sigma^*, i^*) \in \mathsf{List}$ for some i^*. This implies that σ^* was honestly generated. That is, $\sigma^* = (\mathsf{ct}^*, \pi^*)$ where $\mathsf{ct}^* = \Gamma.\mathtt{Enc}([i^*, \cdot]; \mathsf{ek}, \cdot)$. By the correctness of the decryption algorithm $\Gamma.\mathtt{Dec}$, except with negligible probability, $\Gamma.\mathtt{Dec}(\mathsf{ct}; \mathsf{dk}) = (i^*, \cdot)$. Thus, $Z' = i^*$ and $(m^*, \sigma^*, Z') \in \mathsf{List}$, except with negligible probability.

5. $Z''' \neq X^*$:
   As before, since $\sum.\mathtt{Sign}(m^*; \mathsf{sk}_{X^*}, r)$ is honestly generated and $\mathsf{ok}_{Y^*} = \mathsf{dk}$, from the correctness of the decryption algorithm $\Gamma.\mathtt{Dec}$, except with negligible probability, $Z''' = X^*$.

$\square$