Remote Registration of Multiple Authenticators

Yongqi Wang
yxw1112@alumni.bham.ac.uk
University of Birmingham
Birmingham, UK

Thalia Laing
thalia@hp.com
HP Security Lab
Bristol, UK

José Moreira
jose.moreira.sanchez@valory.xyz
Valory AG
Zug, Switzerland

Mark D. Ryan
m.d.ryan@bham.ac.uk
University of Birmingham
Birmingham, UK

ABSTRACT

User authentication with discrete authenticators, such as YubiKeys, is becoming increasingly popular. The authenticators can be external or on-device. They work using challenge-response protocols and public key cryptography. Multiple accounts can be associated with each authenticator. Compared with other forms of authentication, this approach has advantages in security and usability. There are, however, significant limitations which persist. In particular, if users possess only one authenticator, they lack resilience to loss and malfunction. On the other hand, if they possess multiple authenticators, they lack practical solutions to keep authenticators synchronised. In this paper, we present three solutions which combine the usability of a single authenticator with the resilience of multiple authenticators. We also present two key derivation functions which are used in our solutions. All three solutions maintain core security and privacy properties found in existing systems. Meanwhile, each solution provides additional value in different use cases. The security of our solutions is analysed using ProVerif.

CCS CONCEPTS

• **Security and privacy → Key management**; **Authentication**.

KEYWORDS

Entity authentication; discrete authenticators; usability; resilience.

ACM Reference Format:
Yongqi Wang, Thalia Laing, José Moreira, and Mark D. Ryan. 2024. Remote Registration of Multiple Authenticators. In *Proceedings of the Fourteenth ACM Conference on Data and Application Security and Privacy (CODASPY '24), June 19–21, 2024, Porto, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3626232.3653273

1 INTRODUCTION

User authentication is required to secure online services. An increasingly popular approach is to use discrete authenticators, with challenge-response protocols and public-key cryptography, to establish a channel between a *relying party* (RP) and an *authenticator*. The Fast IDentity Online (FIDO) standards are a prominent example of this approach [6, 15].

In this context, RPs are online services which register and authenticate users. Authenticators are components which generate credentials, store credentials, and provide authentication outputs. Authenticators can be embedded in general purpose devices like laptops or smartphones; they can also take the form of external security keys.

Compared to other forms of authentication, such as passwords, discrete authenticators and challenge-response protocols provide advantages in security and usability [13]. There are, however, significant limitations which impede the adoption of these methods and the move away from passwords [22].

In particular, users currently have limited options for managing their authenticators. The basic options are summarised in Table 1. Here, *usability* refers to the work required to register and authenticate. This includes the work required to carry authentication devices or access their physical location. *Resilience* refers to the ability to maintain functionality when something goes wrong; for example, when an authenticator is lost or broken.

To use the form of authentication we are discussing, users must possess at least one authenticator. This authenticator should be readily available to allow users to access their accounts, and create new accounts, at different times and in different locations. In practice, this authenticator is often an object which the user carries on their person. However, with only one authenticator, resilience is limited. If the authenticator is lost or broken, the user will either lose access to their accounts or they will need to carry out burdensome and potentially insecure recovery processes for each account [5]. Therefore, users are advised to possess more than one authenticator [20].

If users possess more than one authenticator, they must decide how to manage these authenticators. Users can either carry additional authenticators on their person or they can store them in a safe location. Currently, both options have significant disadvantages.

If users carry additional authenticators on their person, these authenticators can be registered as soon as a new account is created. However, usability is limited in other ways. Fundamentally, it is inconvenient to carry more than one authenticator for the sake of redundancy. Users are not expected to do this for other necessary items such as mobile phones and house keys. Moreover, by carrying

CODASPY '24, June 19–21, 2024, Porto, Portugal
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0421-5/24/06.
https://doi.org/10.1145/3626232.3653273

Table 1: The trade-off between usability and resilience

Option	Usability	Resilience
1. Possess only one authenticator	Good	Limited
2. Possess two or more authenticators and carry additional authenticators	Limited	Limited
3. Possess two or more authenticators and store back-ups in a safe location	Limited	Good

additional authenticators, resilience is not necessarily improved; two authenticators, such as a security key and a mobile phone, carried in the same coat or bag, can easily be lost at once.

If users store additional authenticators in a safe location, a subset of authenticators is likely to be unavailable when new accounts are established. Periodically, users will need to register these authenticators with recently established accounts. Currently, this requires users to access the physical location of each authenticator. Realistically, many users will forget or neglect to perform these additional registrations. There is also an inevitable delay, between the registration of the original authenticator and subsequent authenticators, during which the original authenticator can be lost or broken.

The problem described here stems from the fact that, with current solutions, each authenticator needs to be physically present at the time of registration. This helps to ensure that registration is secure; however, it is not a necessary condition of security. In this paper, we present solutions which allow several authenticators to be registered when only one authenticator is physically present. In other words, users only need to have one of their authenticators with them to register multiple authenticators. Additional authenticators can be kept offline and in safe locations which are not accessed regularly.

We recognise that authentication occurs in myriad use cases. Moreover, this list of use cases is likely to increase with the proliferation of services and devices. Therefore, we have chosen to present three solutions rather than a single solution. Each solution solves the fundamental problem described above and certain solutions are more suitable for certain use cases. This is discussed in Sec. 4.

We also present two key derivation functions which are used in our solutions. The first allows each authenticator to derive a key pair which is known by all authenticators and unique for each RP. The second allows each authenticator to derive a key pair which is unique for itself and for each RP, whilst also enabling each authenticator to derive the public keys of the other authenticators. Further details are provided in Sec. 5.

Contributions. The aim of our work is to improve usability and resilience when using discrete authenticators. We present three solutions which allow several authenticators to be registered, when only one is physically present, whilst maintaining core security and privacy properties of existing solutions. These authenticators can be used individually or as part of a multi-factor system.

We believe that addressing this problem has real-world utility as physical authenticators greatly improve the security of user authentication and the difficulty of registering back-up authenticators is a hurdle to adoption.

We designed our solutions to be simple and intuitive from the user's perspective. This required some careful consideration of options and requirements. For example, we wanted the authenticators to be interchangeable, rather than having "primary" and "secondary" authenticators which would cause confusion. We also avoided additional entities, such as dealers for group keys or trusted parties for recovery, which would be difficult to incorporate into existing frameworks such as FIDO2.

Organisation. In Sec. 2, we give a brief introduction to the signature schemes used in our solutions. In Sec. 3, we discuss our design goals. Our solutions are defined in Sec. 4. Sec. 5 sets out our key derivation schemes. We analyse the security of our solutions, using ProVerif, in Sec. 6. In Sec. 7, we include some discussion points for context. We conclude in Sec. 8.

2 PRELIMINARIES

The core topic of this paper is the problem of registering multiple authenticators securely and conveniently using challenge-response protocols and public-key cryptography. In this section, we focus on the preliminaries which are directly relevant to our work. Our paper touches on other topics, which facilitate our solutions and analysis, such as key derivation functions and ProVerif proofs. We are conscious of well developed literature in these areas. We have refrained from a lengthy discussion of this literature, in this section, but we have included references to pertinent work at appropriate points in the text.

2.1 Registration and Authentication

User authentication with discrete authenticators and challenge-response protocols involves a registration stage and an authentication stage. In both stages, the user needs physical access to an authenticator.

During registration, the user requests that an authenticator be linked with a new or existing account. The RP sends a challenge to the authenticator. The authenticator generates a public-private key pair and signs the challenge with the private key. The signature and the public key are sent to the RP for verification. If valid, the RP stores the public key and links it with the user's account.

During authentication, the user requests access to an account. The RP sends a fresh challenge to the authenticator. The authenticator signs the challenge with the same private key as above. The signature is sent to the RP. The RP retrieves the public key associated with the account to verify the signature. If valid, the RP grants access to the account.

General purpose devices, such as laptops and smartphones, often have authenticators built into them. This may consist of a number of components working together including, for example, a biometric sensor and a hardware anchor. Meanwhile, mobile devices, like smartphones or wearables, and discrete security keys, like the *YubiKey*, can be used as roaming authenticators when accessing accounts on another device.

2.2 FIDO2 Specifications

We have focused on FIDO2 as the most developed example of authentication with discrete authenticators. To make our solutions practical for real-world applications, we have built on the existing framework of FIDO2, rather than starting something entirely new. FIDO2 does not offer a solution to the problem we have identified, nor have we found a solution in the wider literature.

The FIDO Alliance describes itself as "an open industry association with a focused mission: authentication standards to help reduce the world's over-reliance on passwords" [6]. The alliance consists of a group of major technology companies and adjacent organisations.

The latest set of specifications in this area is called FIDO2. Its purpose is "to leverage common devices to easily authenticate to online services in both mobile and desktop environments" [6]. There are two separate specifications which make up FIDO2. The first is the Web Authentication (WebAuthn) specification and the second is the Client-to-Authenticator Protocol (CTAP) specification.

The primary elements involved in FIDO2 are the authenticators, clients, client devices, and relying parties [15]. Authenticators come in different forms with overlapping properties. Their role is to generate public key credentials, authenticate the user, and provide authentication assertions to the relying party [15].

Next, the clients are web browsers which act as an intermediary between the authenticator and the relying party. The client device, meanwhile, is the hardware on which the client is running and the operating system running on that hardware [6]. Finally, relying parties are the providers of web applications which use the Web Authentication API to register and authenticate users [15].

The interaction between authenticator and client uses CTAP [6]. This incorporates existing standards for USB, Lightning, NFC, and Bluetooth. The interaction between client and relying party uses WebAuthn [15]. This is a passwordless challenge-response protocol using public-key cryptography.

2.3 Schnorr Signatures

In our first two solutions, presented in Sec. 4, we use Schnorr signatures [24], defined below. For some security parameter λ, users agree on a cyclic group $\mathbb{G}$ of order q, where $\|q\| = \lambda$, and a generator g. H is a cryptographically secure hash function such that $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$. The set $\mathcal{P} = (\mathbb{G}, q, g)$ denotes the public parameters of the scheme. In the following description, we define the function symbol $\mathsf{Pk}(x) = g^x \pmod q$. We also include modular reduction notation, which we will omit in subsequent sections for simplicity.

$\mathsf{KeyGen}(\mathcal{P}) = (x, y)$ is the key generation algorithm which takes as input the public parameters and produces a private key x and a public key y:

$$x := \text{a uniformly random integer}, 1 \leq x \leq q-1, \tag{1}$$

$$y := \mathsf{Pk}(x) \overset{\text{def}}{=} g^x \pmod q. \tag{2}$$

$\mathsf{Sign}(x, m) = \sigma$ is the signing algorithm which takes as input a private key x and a message m and produces the signature σ as follows:

$$\begin{aligned} r &:= \text{a random integer}, 1 \leq r \leq q-1, \\ R &:= g^r \pmod q, \\ c &:= H(y \parallel R \parallel m), \\ s &:= r - cx \pmod q, \\ \sigma &:= (c, s). \end{aligned}$$

$\mathsf{Verify}(y, m, \sigma) = v$ is the signature verification algorithm which takes as input a public key y, a message m, and a signature $\sigma = (c, s)$, and produces the verdict v as follows:

$$\begin{aligned} &\hat{R} := g^s\, y^c \pmod q, \\ &\hat{c} := H(y \parallel \hat{R} \parallel m), \\ &\text{If } \hat{c} = c, \text{ then } v := \text{Accept}, \\ &\text{else } v := \text{Reject}. \end{aligned}$$

2.4 Ring Signatures

Our third solution uses ring signatures which were formalised in 2001 by Rivest et al. [23]. They are a means of signing information such that others can verify the authenticity of the information without revealing the identity of the signer. The signer uses two or more public keys, and one corresponding private key, to create a ring signature over some information. The signer does not require help or approval from the owners of the other public keys. Given a list of the public keys, and the ring signature, the RP can verify that one of the corresponding private keys was used to create the signature. However, they cannot determine which public key corresponds to this private key.

The ring signature presented by Rivest et al. is based on RSA signatures. In 2002, Abe et al. presented a more efficient 1-out-of-n ring signature based on Schnorr signatures [4]. To our knowledge, no subsequent scheme offers appreciable improvements in efficiency. Moreover, Schnorr signatures are closely related to the widely used EdDSA. Thus, we have used Abe's ring signature in our solution and developed a compatible key derivation process, shown in Sec. 5.

Abe et al.'s ring signature [4] is as follows. Key generation results in a collection of private keys, $x_1, \ldots, x_n$, and their corresponding public keys, $y_1, \ldots, y_n$. Signing is executed by any participant i by taking their private key x_i, and all public keys in the ring, $y_1, \ldots, y_n$, as input. For convenience, we denote the array of public keys as $L = (y_1, \ldots, y_n)$. During verification, the equivalence holds if the signature was created using an x_i among $x_1, \ldots, x_n$. The verification process is the same, regardless of which x_i was used.

$\mathsf{KeyGen}(\mathcal{P}) = (x_i, y_i)$ takes as input public parameters $\mathcal{P}$ and outputs a private key x_i, and public key y_i, as defined by (1)–(2). This algorithm is executed n times.

$\mathsf{RingSign}(x_i, L, m) = \sigma$ is the ring signing algorithm which takes as input a private key x_i, a list of public keys $L = (y_1, \ldots, y_n)$, and a message m, and produces the ring signature σ as follows:

$$\begin{aligned} c_1, \ldots, c_{i-1}, & \\ c_{i+1}, \ldots, c_n &:= \text{random integers}, 1 \leq c_j \leq q-1, \\ r &:= \text{a random integer}, 1 \leq r \leq q-1, \\ R &:= g^r y_1^{c_1} \cdots y_{i-1}^{c_{i-1}} \cdot y_{i+1}^{c_{i+1}} \cdots y_n^{c_n}, \\ c &:= H(L \parallel R \parallel m), \\ c_i &:= c - c_1 - \cdots - c_{i-1} - c_{i+1} - \cdots - c_n, \\ s &:= r - c_i x_i, \\ \sigma &:= (c_1, \ldots, c_n, s). \end{aligned}$$

$\mathsf{RingVerify}(L, m, \sigma) = v$ is the ring signature verification algorithm which takes as input a list of public keys $L = (y_1, \ldots, y_n)$, a message m, and a ring signature $\sigma = (c_1, \ldots, c_n, s)$, and produces the verdict v as follows:

$$\hat{c} := H(L \parallel g^s\, y_1^{c_1} \cdots y_n^{c_n} \parallel m),$$
$$\text{If } \hat{c} = c_1 + \cdots + c_n, \text{ then } v := \text{Accept},$$
$$\text{else } v := \text{Reject}.$$

3 DESIGN GOALS

The solutions we present in Sec. 4 satisfy the following design goals. To overcome the problem described in Sec. 1, without requiring a significant increase in resources, we have the following goals.

G1 **Convenient registration of back up authenticators.** Multiple authenticators can be registered with only one registration session and one authenticator physically present at the time of registration. All other authenticators can be kept offline and in remote locations.

G2 **Convenient storage of back up authenticators.** Apart from an introduction phase (discussed in Sec. 4) in which the authenticators establish a shared secret between themselves, the authenticators do not need to communicate with each other; thus, back up authenticators can be stored in safe locations which are not accessed regularly.

G3 **No significant increase in authenticator requirements.** Compared to existing implementations, our solutions do not require a significant increase in resources. All cryptographic operations performed by the authenticators are efficient and widely implemented. In addition, keys associated with particular RPs can be derived during registration and authentication. This minimises the information that each authenticator needs to store.

G4 **No significant increase in RP requirements.** Compared to existing implementations of RPs, our solutions do not require a significant increase in resources. All cryptographic operations performed by the RPs are efficient and widely implemented.

G5 **No additional entities.** Similar to the FIDO2 Standard [6, 15], no additional entities are required as intermediaries or trusted third parties. In other words, our goal is to avoid introducing extra entities to those required in FIDO2, whilst solving the problem which we have identified.

In addition, we include a number of properties from FIDO2 which are important to maintain [7]. (Note, FIDO2 has other properties which are beyond the scope of this paper.) For our purposes, the key properties are as follows.

G6 **Secure hardware-based authentication.** Only those with physical access to a registered authenticator can authenticate. Another factor, such as a password or a biometric, may be added for local authentication (of the user to the authenticator) or as a second factor for the RP.

G7 **Secure registration of authenticators.** Authenticators can be registered when an account is created. They can also be registered during a session in which the user has already authenticated with a registered authenticator. Authenticators cannot be registered in any other instance.

G8 **Interchangeable authenticators.** For convenience and resilience, the user can designate a set of authenticators such that each authenticator in the set has the same functionality and can be used interchangeably. The loss or destruction of a subset of authenticators does not prevent the remaining authenticators from functioning.

G9 **Privacy across RPs.** A user's interactions with an RP cannot be linked to their interactions with any other RP.

The property of privacy across RPs (unlinkability between different RPs) is motivated by existing implementations such as FIDO2 (see goal SG-4 in [7]).

Two of our solutions, described in the following section, also provide a complementary property which is privacy across authenticators. This is useful when the user does not want to disclose to the RP which devices they are using. In other words, valid signatures from different authenticators, authenticating to the same account, cannot be distinguished by the RP. Examples of information hungry RPs are abundant and the pattern of authenticator use can be a valuable data point for advertisers and other entities. For example, the RP may be able to infer a lost device, or the location of a user, based on authenticator use.

4 SOLUTIONS

In this section, we present three solutions to the problem introduced in Sec. 1. We call these **Duplicate**, **Proxy**, and **Ring** Authenticators. All three solutions satisfy the design goals, presented in Sec. 3, and each relies on a key derivation algorithm, presented in Sec. 5. Differences between the solutions, which are advantageous in different scenarios, are outlined in Table 2.

Duplicate Authenticators. Our first solution, Duplicate Authenticators, requires the least storage. In Proxy and Ring, each authenticator must store a public key for each other authenticator. Thus, the number of public keys stored, and the amount of memory required, increases linearly with the number of authenticators. Duplicate Authenticators break this relationship by having a common key for all authenticators; therefore, each authenticator is only required to store one public key. This does, however, come at a cost to security: if one authenticator is compromised in the duplicate solution, all authenticators are compromised. This is not true for the Proxy and Ring solutions.

Minimising memory is particularly valuable when authenticators are embedded in everyday objects. In addition to being secure and usable, these implementations need to be small, simple, and inexpensive. These qualities will also increase the accessibility of physical authenticators for wider user groups.

Furthermore, Duplicate Authenticators offer a high level of privacy since valid signatures from different authenticators, authenticating to the same account, cannot be distinguished by the RP.

Ring Authenticators. Similar to Duplicate Authenticators, Ring Authenticators offer a high level of privacy where valid signatures from different authenticators cannot be distinguished by the RP. In addition, Ring Authenticators allow for unique key pairs on each authenticator. This offers security benefits over the Duplicate solution; each authenticator can generate its own key pair and, if one authenticator is compromised, the others maintain their security. This solution is valuable in scenarios where security and privacy are paramount, but memory is less restricted.

Table 2: Differences between the three solutions

Property	Duplicate	Proxy	Ring
The authenticators can be registered and revoked independently (see Sec. 7.2).	✗	✓	✗
The key pairs used for authentication are unique to each authenticator (and to each RP).	✗	✓	✓
If the shared secret k is leaked, the security of the private keys would be unaffected.	✗	✓	✓
Given two valid signatures from two authenticators, an RP cannot distinguish between them (unless other information is included).	✓	✗	✓
The authenticators do not need to store the public keys of other authenticators.	✓	✗	✗

Finally, *Proxy Authenticators* are particularly beneficial when the set of authenticators often changes. Each authenticator can be registered and revoked easily and independently. This is because the public keys are independent, which is not the case with Duplicate or Ring Authenticators. In particular, there is no need to contact each RP that the incoming or outgoing authenticator is registered with. In a world where the average user has more than 100 accounts [25], this can significantly reduce the workload required by the user.

Also, similar to Ring Authenticators, every authenticator has its own unique key. Thus, Proxy Authenticators offer a convenient, secure solution when there is no need to maximise privacy (as in Ring and Duplicate) or minimise storage (as in Duplicate). We believe this is applicable to many everyday authentication scenarios.

Each solution we present consists of three stages: introduction, registration, and authentication. The introduction stage is the only stage which requires more than one authenticator to be present. It is carried out during an initial setup where two or more authenticators form a set by establishing a shared high entropy secret k. It is repeated only when a new authenticator is added to the set. The registration stage requires only one authenticator to be present for multiple authenticators to be registered. It is carried out each time the user creates a new account with an RP. It is repeated when the user wants to add to the set of authenticators registered with an RP. The authentication stage requires one authenticator to be present. It is carried out each time the user is required to authenticate before accessing one of their accounts.

The sequences below are presented for the case of two authenticators, A_1 and A_2. In each case, A_1 registers both authenticators, without A_2 being present during registration, and A_2 authenticates later on. It is straightforward to generalise the sequences for n interchangeable authenticators.

Note that the three solutions require the commonplace security practice of tagging the registration and authentication protocols [3, 10]. In this case, we use the constant values "REG" and "AUTH". If this security measure is not implemented, the adversary could mix-and-match messages from both protocols, breaking the expected security properties (see Sec. 6.2). We recall that the security of a tagged protocol does not imply the security of its untagged version.

Solution 1: Duplicate Authenticators. In this solution, A_1 and A_2 each derive the same key pair which is unique to the relying party RP_j. That is, $(x_{1j}, y_{1j}) = (x_{2j}, y_{2j})$. In other words, A_1 and A_2 are duplicates. A_1 registers the public key with the RP. Subsequently, A_2 can authenticate using the private key. The protocols for this solution are depicted in Fig. 1, where $\mathsf{Der}(k, RP_j)$ is used to generate the key pairs; this is defined in Sec. 5. Sign and Verify are the signature and verification algorithms defined in Sec. 2.3.

Solution 2: Proxy Authenticators. In this solution, A_1 and A_2 have different primary key pairs, (x_1, y_1) and (x_2, y_2), respectively. These key pairs are used to derive a unique key pair per authenticator and RP. First, A_1 uses $\mathsf{DerMulti}(x_1, (y_1, y_2), k, RP_j)$, defined in Sec. 5, to derive its own private key x_{1j} and the list of public keys $L_j = (y_{1j}, y_{2j})$ linked to RP_j. Then, it registers L_j with the RP. In other words, A_1 nominates A_2 as a proxy. Subsequently, A_2 derives its own private key x_{2j} in a similar way, and creates a valid signature as $\mathsf{Sign}\big(x_{2j}, (\mathrm{AUTH}, m_2)\big)$ to authenticate the user. The protocols for this solution are depicted in Fig. 2. Again, Sign and Verify are the algorithms defined in Sec. 2.3.

Solution 3: Ring Authenticators. Similarly, in this solution, A_1 and A_2 have primary key pairs that are used to derive unique key pairs per authenticator and RP. The key derivation process is the same as for proxy authenticators above. A_1 registers $L_j = (y_{1j}, y_{2j})$ with the RP, nominating A_2 as a member of the ring. Subsequently, A_2 can derive its own private key x_{2j} and the list of public keys L_j to create a valid ring signature using $\mathsf{RingSign}\big((x_{2j}, L_j, (\mathrm{AUTH}, m_2)\big)$. The protocols for this solution are depicted in Fig. 3. RingSign and RingVerify are the ring signature and verification algorithms defined in Sec. 2.4.

Privacy Properties. All three solutions satisfy the standard privacy property which is featured in FIDO2; namely, privacy across relying parties. This follows immediately from the fact that a unique public key is used for each RP.

Duplicate and Ring Authenticators also satisfy a complementary privacy property which is privacy across authenticators. For Duplicate Authenticators, this property follows from the fact that A_1 and A_2 interact with the RP using identical protocols and the public keys for both authenticators, y_{1j} and y_{2j}, are identical. Therefore, there is no information with which the RP can distinguish between A_1 and A_2. This property generalises immediately to the case of n authenticators with n identical public keys.

For Ring Authenticators, the property of privacy across authenticators follows directly from the basic properties of a ring signature. Given a ring signature and the corresponding list of public keys, L_j, the RP cannot determine which public key corresponds to the private key which was used to generate the signature. In other words, from the perspective of the RP, each ring member is equally probable to have been the one who created the signature.

5 KEY DERIVATION

In this section, we present two key derivation algorithms, Der and DerMulti, which allow authenticators to derive related keys used in our solutions. Der, used in Solution 1, allows the authenticators to derive a unique key pair per RP by using the shared secret k as the seed. This key pair is known by all the authenticators. DerMulti, used in Solutions 2 and 3, allows the authenticators to derive a

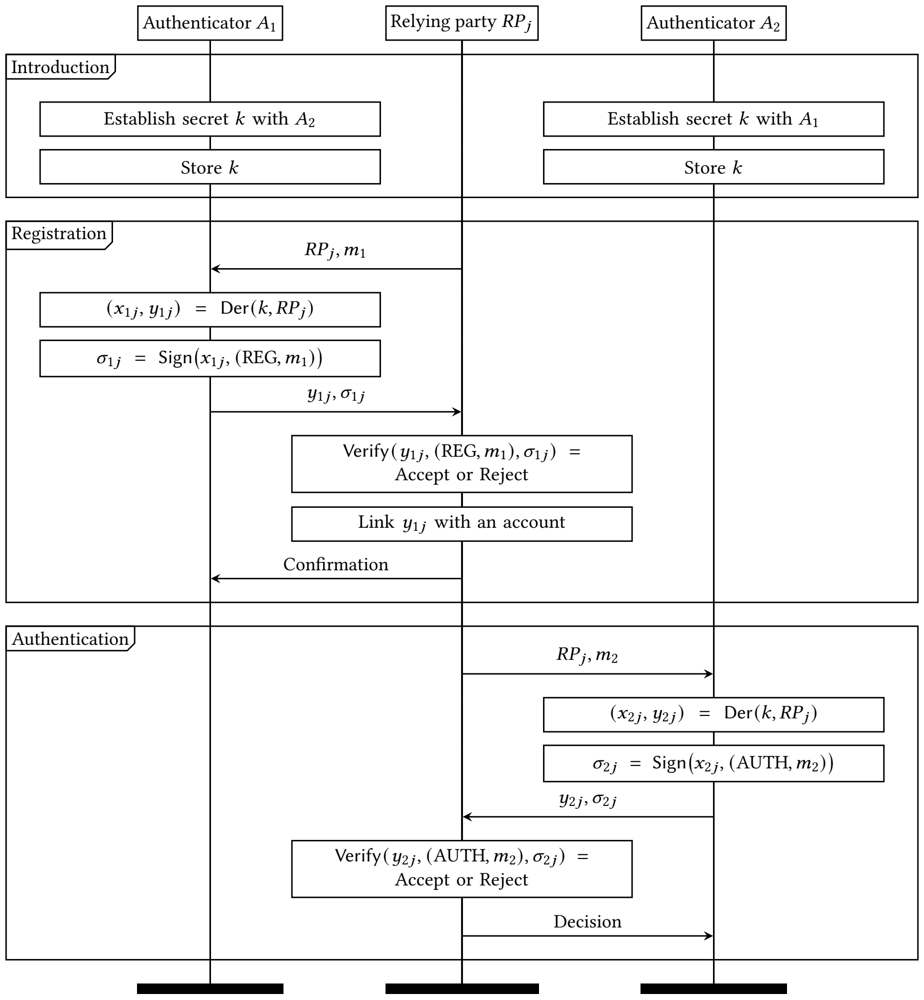
Authenticator A_1
Relying party RP_j
Authenticator A_2
Introduction
Establish secret k with A_2
Store k
Establish secret k with A_1
Store k
Registration
RP_j, m_1
$(x_{1j}, y_{1j}) = \mathsf{Der}(k, RP_j)$
$\sigma_{1j} = \mathsf{Sign}(x_{1j}, (\mathrm{REG}, m_1))$
y_{1j}, σ_{1j}
$\mathsf{Verify}(y_{1j}, (\mathrm{REG}, m_1), \sigma_{1j}) =$ Accept or Reject
Link y_{1j} with an account
Confirmation
Authentication
RP_j, m_2
$(x_{2j}, y_{2j}) = \mathsf{Der}(k, RP_j)$
$\sigma_{2j} = \mathsf{Sign}(x_{2j}, (\mathrm{AUTH}, m_2))$
y_{2j}, σ_{2j}
$\mathsf{Verify}(y_{2j}, (\mathrm{AUTH}, m_2), \sigma_{2j}) =$ Accept or Reject
Decision

Figure 1: Message sequence for Solution 1: Duplicate Authenticators

unique key pair per RP *and* per authenticator by using both the shared secret k and a unique, per-authenticator private key x_i as seeds. Moreover, DerMulti derives the public keys of the other authenticators that correspond to the same RP. Der and DerMulti are constructed to satisfy the following requirements:

- R1 **Unique key pairs.** The derived key pairs are unique for each RP. Using the same public key across different RPs can lead to vulnerabilities in privacy. This would conflict with our design goals in Sec. 3.
- R2 **Repeatable key derivation.** Key pairs can be derived deterministically and on demand. This minimises information that authenticators must store.
- R3 **Secure private keys.** Each authenticator in the set can derive exactly one private key for each RP. No entity outside of the set can compute any private keys.
- R4 **Mutually derived public keys.** In addition to their own public key, each authenticator can derive the public keys of other authenticators in the set.
- R5 **Privacy preserving public keys.** Other than authenticators in the set, no other entity can derive the public keys which are used for authentication. Otherwise, they would be able to link public keys across RPs.
- R6 **Compatibility with chosen signature schemes.** Finally, the derived key pairs should be compatible with the appropriate signature scheme for each solution in Sec. 4.

The two key derivation algorithms executed by the authenticators rely on a shared secret k. Suppose we want to generate keys for authenticator i and relying party RP_j.

$\mathsf{Der}(k, RP_j) = (x_{ij}, y_{ij})$ is a key derivation algorithm, which takes as input a shared secret k and an RP identifier RP_j, and produces a private key x_{ij} and a public key y_{ij} as follows:

$$x_{ij} := H(k \parallel RP_j),$$
$$y_{ij} := \mathsf{Pk}(x_{ij}) \stackrel{\text{def}}{=} g^{x_{ij}}.$$

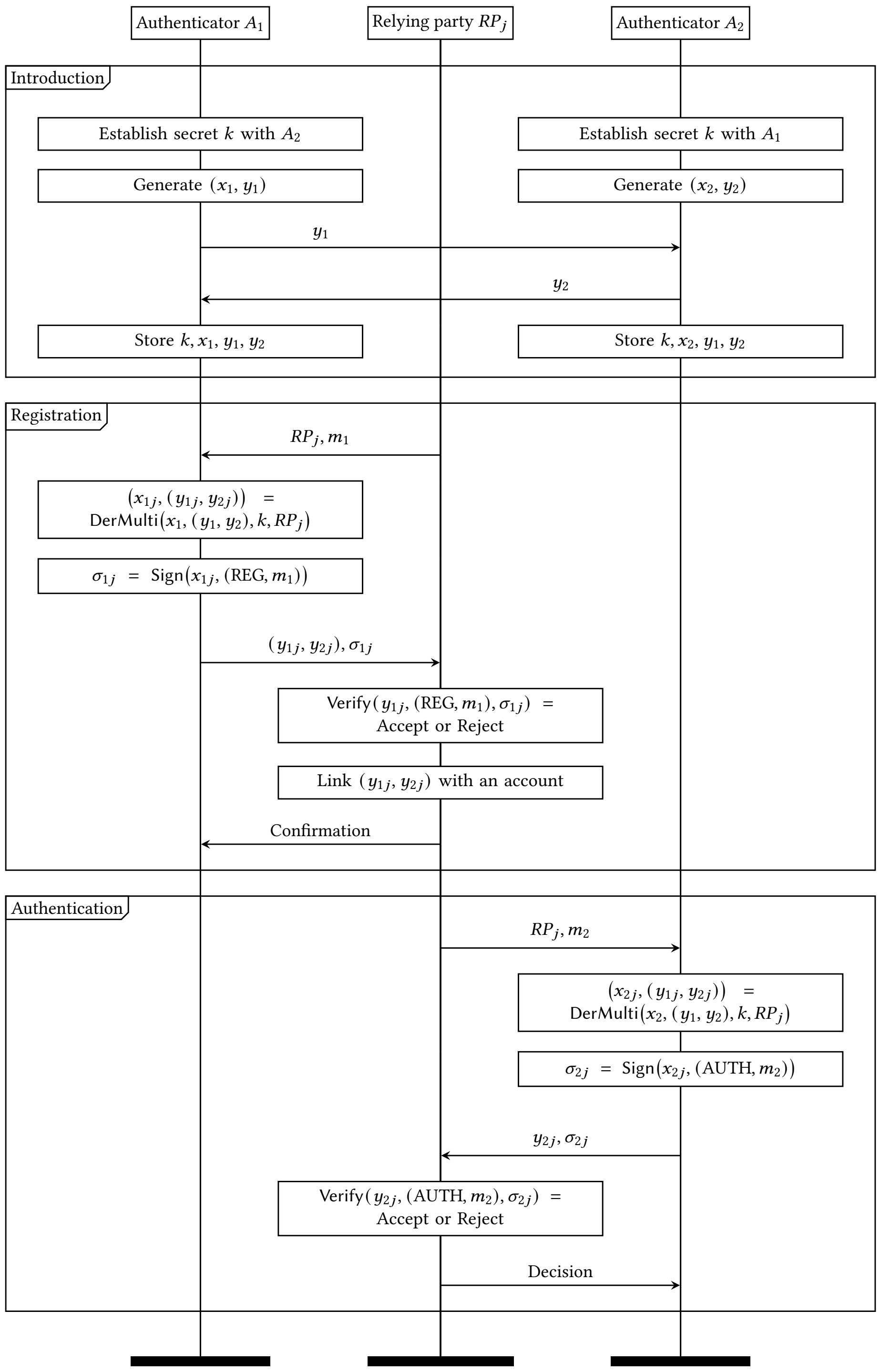
Authenticator A_1
Relying party RP_j
Authenticator A_2
Introduction
Establish secret k with A_2
Establish secret k with A_1
Generate (x_1, y_1)
Generate (x_2, y_2)
y_1
y_2
Store k, x_1, y_1, y_2
Store k, x_2, y_1, y_2
Registration
RP_j, m_1
$(x_{1j}, (y_{1j}, y_{2j})) =$ DerMulti$(x_1, (y_1, y_2), k, RP_j)$
$\sigma_{1j} =$ Sign$(x_{1j}, (\text{REG}, m_1))$
$(y_{1j}, y_{2j}), \sigma_{1j}$
Verify$(y_{1j}, (\text{REG}, m_1), \sigma_{1j}) =$ Accept or Reject
Link (y_{1j}, y_{2j}) with an account
Confirmation
Authentication
RP_j, m_2
$(x_{2j}, (y_{1j}, y_{2j})) =$ DerMulti$(x_2, (y_1, y_2), k, RP_j)$
$\sigma_{2j} =$ Sign$(x_{2j}, (\text{AUTH}, m_2))$
y_{2j}, σ_{2j}
Verify$(y_{2j}, (\text{AUTH}, m_2), \sigma_{2j}) =$ Accept or Reject
Decision

Figure 2: Message sequence for Solution 2: Proxy Authenticators

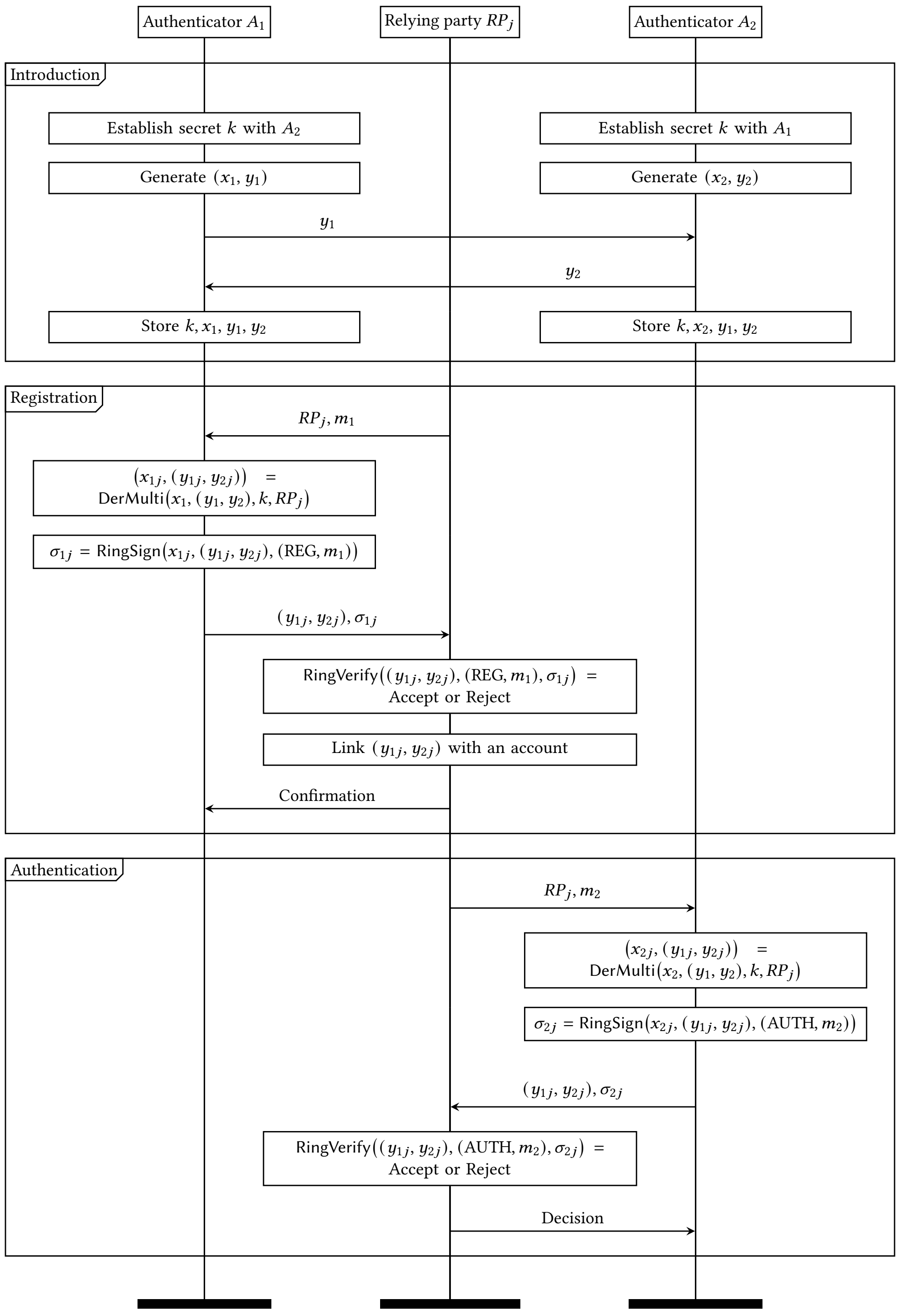
Authenticator A_1
Relying party RP_j
Authenticator A_2
Introduction
Establish secret k with A_2
Establish secret k with A_1
Generate (x_1, y_1)
Generate (x_2, y_2)
y_1
y_2
Store k, x_1, y_1, y_2
Store k, x_2, y_1, y_2
Registration
RP_j, m_1
$(x_{1j}, (y_{1j}, y_{2j})) = \text{DerMulti}(x_1, (y_1, y_2), k, RP_j)$
$\sigma_{1j} = \text{RingSign}(x_{1j}, (y_{1j}, y_{2j}), (\text{REG}, m_1))$
$(y_{1j}, y_{2j}), \sigma_{1j}$
$\text{RingVerify}((y_{1j}, y_{2j}), (\text{REG}, m_1), \sigma_{1j}) =$ Accept or Reject
Link (y_{1j}, y_{2j}) with an account
Confirmation
Authentication
RP_j, m_2
$(x_{2j}, (y_{1j}, y_{2j})) = \text{DerMulti}(x_2, (y_1, y_2), k, RP_j)$
$\sigma_{2j} = \text{RingSign}(x_{2j}, (y_{1j}, y_{2j}), (\text{AUTH}, m_2))$
$(y_{1j}, y_{2j}), \sigma_{2j}$
$\text{RingVerify}((y_{1j}, y_{2j}), (\text{AUTH}, m_2), \sigma_{2j}) =$ Accept or Reject
Decision

Figure 3: Message sequence for Solution 3: Ring Authenticators

***Security of the* Der *algorithm*.** Suppose H is a cryptographic hash function, k is secret information, and RP_j is a salt value. By definition, $H(k \parallel RP_j)$ is identical to a one-step key derivation function which is included in the international standard ISO/IEC 11770-6 and denoted OKDF1 [19]. The security of the private key, $x_{ij} := H(k \parallel RP_j)$, follows directly from the security of OKDF1. In other words, x_{ij} is a uniformly random integer. Therefore, $y_{ij} := g^{x_{ij}}$ is a valid public key for a Schnorr Signature, as defined in Sec 2.3, and (x_{ij}, y_{ij}) is a valid key pair for a Schnorr Signature.

$\mathsf{DerMulti}(x_i, L, k, RP_j) = (x_{ij}, L_j)$ is a key derivation algorithm, which takes as input a private key x_i, a list of public keys $L = (y_1, \dots, y_n)$, a shared secret k and an RP identifier RP_j, and outputs a private key x_{ij}, and a list of public keys $L_j = (y_{1j}, \dots y_{nj})$. DerMulti is applied by each authenticator A_i, using the authenticator's private key x_i. The correctness assumption is that $y_i = \mathsf{Pk}(x_i)$ holds for the input arguments. For convenience, we introduce the function symbols DerSk and DerPk:

(1) Compute own private key:

$$k_{ij} := H(\mathsf{Pk}(x_i) \parallel k \parallel RP_j),$$

$$x_{ij} := \mathsf{DerSk}(x_i, k, RP_j) \overset{\text{def}}{=} x_i \cdot k_{ij}.$$

(2) Compute all authenticators' public keys:

$$\text{for } 1 \le i' \le n:$$

$$k_{i'j} := H(y_{i'} \parallel k \parallel RP_j)$$

$$y_{i'j} := \mathsf{DerPk}(y_{i'}, k, RP_j) \overset{\text{def}}{=} y_{i'}^{k_{i'j}}.$$

***Security of the* DerMulti *algorithm*.** For $1 \le i, j \le n$, suppose x_i are uniformly random integers, $y_i = g^{x_i}$ are Schnorr Signature public keys, H is a cryptographic hash function, k is secret information, and the remaining inputs of $H(y_i \parallel k \parallel RP_j)$ are salt values. By definition, $H(y_i \parallel k \parallel RP_j)$ is identical to OKDF1 as specified in ISO/IEC 11770-6 [19]. The security of $k_{ij} := H(y_i \parallel k \parallel RP_j)$ follows directly from the security of OKDF1. In other words, k_{ij} are uniformly random integers for $1 \le i, j \le n$. Therefore, $x_{ij} := x_i \cdot k_{ij}$, the product of two uniformly random integers, is also a uniformly random integer.

Hence, for $1 \le i, j \le n$, $y_{ij} := y_i^{k_{ij}} = g^{x_i k_{ij}} = g^{x_{ij}}$ is a valid public key for a Schnorr Signature, as defined in Sec 2.3, and (x_{ij}, y_{ij}) is a valid key pair for a Schnorr Signature. The security of the new key pair, (x_{ij}, y_{ij}), is equivalent to the security of the original key pair, (x_i, y_i) since the secret x_i is required to compute $x_{ij} := x_i \cdot k_{ij}$.

Subsequently, it is straightforward to confirm that both Der and DerMulti satisfy the requirements above. R1 follows from the uniqueness of RP_j and due to the collision resistance property of cryptographic hash functions. R2 holds because the hash function H is deterministic and all the inputs are constants. For Der, R3 holds because k is only known to the authenticators in the set and, due to the security of the underlying signature scheme, the public key will not reveal any information about the private key. For DerMulti, R3 depends on the fact that only authenticators in the set can compute k_{ij}, as only they know k. Then each private key, x_i, is only known to one authenticator, thus the output of the multiplication $x_i \cdot k_{ij}$, which is kept private, can only be computed by the one authenticator. For Der, R4 holds immediately since the derivation is deterministic, thus all authenticators will compute the same key pair (x_{ij}, y_{ij}). For DerMulti, R4 follows from the definition of $y_{i'j}$ for which the inputs are known to all authenticators in the set and because the derivation is deterministic. R5 holds because k is a secret shared by only the authenticators in the set, thus only the authenticators in the set can derive k_{ij}. Non-authenticators will then be unable to compute y_{ij} without knowledge of k_{ij}. Finally, R6 follows by definition.

Table 3: Notation

Symbol	Description
A_i	Authenticator $i \in \{1, \dots, n\}$.
RP_j	Unique identifier of relying party j.
k	A strong secret shared by all the authenticators.
L	A list of public keys; see Sec. 2.4.
L_j	A list of public keys associated with RP_j; see Sec. 5.
m	A challenge sent by the relying party.
(x_i, y_i)	A primary key pair belonging to A_i.
(x_{ij}, y_{ij})	A derived key pair belonging to A_i and associated with RP_j.
REG	Registration tag.
AUTH	Authentication tag.

Note that the algorithm Der does not derive keys based on any authenticator-specific secret, hence $(x_{ij}, y_{ij}) = (x_{i'j}, y_{i'j})$ for all i and i'. On the other hand, DerMulti requires that each authenticator has a unique, primary key pair (x_i, y_i) that must have been generated in advance. Then, Authenticator i uses that key pair to derive secondary key pairs (x_{ij}, y_{ij}) that are unique to Authenticator i and relying party RP_j. Any Authenticator i is able to obtain its own secondary key pair (x_{ij}, y_{ij}) through the usage of DerMulti, but can also obtain the secondary public keys $y_{i'j}$ of the other authenticators for that relying party RP_j.

For all x_i, k, RP_j, we have

$$\mathsf{DerPk}(\mathsf{Pk}(x_i), k, RP_j) = \mathsf{Pk}(\mathsf{DerSk}(x_i, k, RP_j)), \tag{3}$$

which enables any authenticator to derive the public keys of the other authenticators.

In addition, unlike for Der, the shared secret k is not used to generate any keys directly in DerMulti. Instead, it is included to prevent anyone other than the legitimate authenticators from generating and linking the secondary public keys $y_{1j}, \dots, y_{nj}$ associated with RP_j. If k is leaked, the security of the private keys would be unaffected. However, there would be an impact on privacy since an adversary could link the secondary public keys.

6 MODELLING AND VERIFICATION OF SECURITY PROPERTIES

We have developed models for the three solutions (Duplicate, Proxy and Ring) in the symbolic model of cryptography using the automated protocol verifier ProVerif [11, 12]. Moreover, we have developed a macro language to adapt the ProVerif syntax to our needs.

ProVerif receives as input a protocol specified in the applied pi-calculus [2] and a set of security properties to query, specified in a syntax equivalent to a first-order logic formula. ProVerif works under the assumption of a Dolev-Yao adversary [17], which is able to intercept, replay, delay, suppress and modify messages (subject to the algebra on defined primitives). We recall that ProVerif is sound but not complete, because the verification of security properties is undecidable for an unbounded number of protocol sessions and message space [1, 8, 18]. However, if it claims that a protocol satisfies a security property, then the property is actually satisfied. When the tool cannot prove a property, it tries to reconstruct an attack, i.e., an execution trace of the protocol that falsifies that property. We refer the reader, e.g., to [10–12, 16, 17] for a further discussion on the topic of formal verification in the symbolic model and ProVerif.

We have considered a number of security properties detailed below in order to prove both the design goals (G1-G9 from Sec. 3) and the key derivation design requirements (R1-R6 from Sec. 5). ProVerif is able to prove that all three of our solutions meet all the defined security properties. We remark that some of the goals or requirements are either not verifiable in the symbolic model or not expressible as security properties.

The repository with the models is available at [26], and we provide the most important details for them below.

6.1 Equational Theories

We require two equational theories for the three solutions; namely, signatures and ring signatures, as described in Sec. 2. To accommodate probabilistic signatures, it is customary to define an additional function symbol that handles the randomness. Therefore, for the case of signatures from Sec. 2.3, the equational theory consists of the function symbols Pk/1, Sign/2, InternalSign/3, and Verify/3, so that the ProVerif code for the signing function looks as follows:

$$\texttt{letfun}\ \mathsf{Sign}(x, m)\ =\ \texttt{new}\ r;\ \mathsf{InternalSign}(x, m, r)\,.$$

Hence, the matching rewrite rule in the destructor [12, Sec. 3.1.1] is

$$\mathsf{Verify}(\mathsf{Pk}(x), m, \mathsf{InternalSign}(x, m, r))\ =\ \texttt{true}\,.$$

For the case of ring signatures from Sec. 2.4, we also define a probabilistic scheme with function symbols Pk/1, RingSign/3, InternalRingSign/4, and RingVerify/3, and a type for the list of public keys built with the data constructor $\mathsf{PkList}(y_1, \dots, y_n)$. Thus,

$$\begin{aligned}&\texttt{letfun}\ \mathsf{RingSign}(x, L, m)\ =\\ &\quad\texttt{new}\ r;\ \mathsf{InternalRingSign}(x, L, m, r)\,.\end{aligned}$$

Since it should not be possible to determine which key in the ring produced a signature, there is the need to define n rewrite rules for the destructor, in order to reflect the fact that any key in the ring is capable of producing a valid signature. That is, for all lists of public keys $L = \mathsf{PkList}(\mathsf{Pk}(x_1), \dots, \mathsf{Pk}(x_n))$, we have

$$\begin{aligned}&\mathsf{RingVerify}(L, m, \mathsf{InternalRingSign}(x_1, L, m, r))\ =\ \texttt{true}\\ &\quad\vdots\\ &\mathsf{RingVerify}(L, m, \mathsf{InternalRingSign}(x_n, L, m, r))\ =\ \texttt{true}\,.\end{aligned}$$

ProVerif does not offer support to expand these collections of rewrite rules for an arbitrary value n passed as a parameter. Hence, we have developed a macro language that handles this operation conveniently. See Sec. 6.3 below.

Finally, we also model the theories defined in Sec. 5. Recall that a given authenticator A_i uses DerSk to derive the secret key associated with itself and RP_j, and it uses DerPk to derive the public keys of all the authenticators for RP_j. Consequently, we define the function symbols DerPk/3 and DerSk/3 and make explicit their relationship in Eq. 3 as a ProVerif equation, which is required so that the tool can complete the verification:

$$\begin{aligned}&\texttt{equation forall}\ x_i,\ k,\ RP_j;\\ &\quad\mathsf{DerPk}(\mathsf{Pk}(x_i), k, RP_j) = \mathsf{Pk}(\mathsf{DerSk}(x_i, k, RP_j))\,.\end{aligned}$$

Note that due to technical constraints of the resolution algorithm, this equation must be defined as presented above. Swapping the LHS and RHS will cause the algorithm to fail.

6.2 Events and Security Properties

In the symbolic model, the adversary is in control of the public channel used by the honest parties to exchange messages. Events are used to annotate the protocol at specific places without changing the semantics of the processes, and to define security properties. Events can be thought of as "local computations" or as mere check points on locations of interest that record object values. We prepend "A_" to events executed at an authenticator process, and "R_" to events executed at an RP process. We follow a naming convention similar to [21] to name the events:

A_Register: Executed before an authenticator outputs a register message, i.e., the list of public keys to register to RP_j.

A_Running: Executed before an authenticator outputs a signed challenge to authenticate against an RP.

R_Register: Executed after an RP receives a registration request.

R_Commit: Executed after an RP validates a signature from an authenticator.

By convention, the parameters in the events follow the order 1) terms referring to the identity of the party where the event is launched, 2) available terms referring to the other party identity, and 3) agreement parameters. For example,

$$\mathsf{A\_Running}(\underbrace{uid,\ \ i,\ \ x_i,\ \ L_j,}_{\text{User and authenticator identity terms}}\ \underbrace{RP_j,}_{\text{RP identity term}}\ \underbrace{p}_{\text{Agreement parameters}}).$$

Of course, not all events make use of all the terms. For example, RPs do not have access to the user identifier uid, the authenticator index i nor the private key x_i, and therefore these parameters are omitted for events executed at the RP side.

Using these events, we can describe a number of high-level security properties. The main focus is to verify correct authentication between any registered authenticator belonging to a user with identifier uid. The notation $\mathsf{Event}(p_1, \dots, p_n)@t$ signifies that event "Event" was executed with parameters $p_1, \dots, p_n$ at time t. We only show the security properties for ring authenticators (duplicate and proxy authenticators follow similarly). To avoid ambiguity, we present the security properties as fully expanded first-order logic formulas, although ProVerif provides a simplified syntax.

Sanity check. It is reachable that all the authenticators belonging to a user uid cast the same list of public keys to register with RP_j:

$$\begin{aligned}&\exists uid, x_1, \dots, x_n, k, RP_j, t_1, \dots, t_n.\\ &\quad\mathsf{A\_Register}(uid, 1, x_1, L_j, RP_j)@t_1 \land\\ &\quad\vdots\\ &\quad\mathsf{A\_Register}(uid, n, x_n, L_j, RP_j)@t_n,\end{aligned}$$

where L_j denotes the list of public keys, derived as

$$L_j = \mathsf{PkList}\big(\mathsf{Pk}(\mathsf{DerSk}(x_1, k, RP_j)), \ldots, \mathsf{Pk}(\mathsf{DerSk}(x_n, k, RP_j))\big). \tag{4}$$

Property 1 (G1, G2, G7, G8, R4). All authenticators of user *uid* register the same list of public keys with RP_j:

$$\begin{aligned}&\forall uid, i, i', x_i, x_{i'}, L_j, L'_j, RP_j, t, t'.\\ &\quad \mathsf{A\_Register}(uid, i, x_i, L_j, RP_j)@t \wedge\\ &\quad \mathsf{A\_Register}(uid, i', x_{i'}, L'_j, RP_j)@t' \quad \Rightarrow L_j = L'_j.\end{aligned}$$

Property 2 (G1, G7, G8, R1, R3). All authenticators of user *uid* register a given list of public keys with at most one RP:

$$\begin{aligned}&\forall uid, i, i', x_i, x_{i'}, L_j, RP_j, RP'_j, t, t'.\\ &\quad \mathsf{A\_Register}(uid, i, x_i, L_j, RP_j)@t \wedge\\ &\quad \mathsf{A\_Register}(uid, i', x_{i'}, L_j, RP'_j)@t' \quad \Rightarrow RP_j = RP'_j.\end{aligned}$$

Property 3 (G1, G2, G7, G8, R4). It is reachable that an authenticator of user *uid* registers all their authenticators, and any other authenticator of *uid* can then log in. We present an instantiation of this property showing that Authenticator 1 registers with RP_j, and all the other authenticators are able to authenticate:

$$\begin{aligned}&\exists uid, x_1, \ldots, x_n, k, RP_j, p_1, \ldots, p_n, t, t_1, \ldots, t_n, t'_1, \ldots, t'_n.\\ &\quad \mathsf{A\_Register}(uid, 1, x_1, L_j, RP_j)@t \wedge\\ &\quad \mathsf{A\_Running}(uid, 1, x_1, L_j, RP_j, p_1)@t_1 \wedge\\ &\qquad \mathsf{R\_Commit}(RP_j, L_j, p_1)@t'_1 \wedge\\ &\qquad \vdots\\ &\quad \mathsf{A\_Running}(uid, n, x_n, L_j, RP_j, p_n)@t_n \wedge\\ &\qquad \mathsf{R\_Commit}(RP_j, L_j, p_n)@t'_n,\end{aligned}$$

where L_j denotes the associated list of public keys as in (4).

ProVerif finds a trace in which *only* Authenticator 1 registers and the others authenticate. Although we could have considered adding restrictions to prevent other authenticators from registering, this was unnecessary since the trace found by ProVerif already has this property. We can establish analogous formulas showing a trace where only Authenticator i registers ($i > 1$) and all others authenticate.

Finally, the last property corresponds to a variation of injective agreement [21, Sec. 2.4] for registered authenticators. This is the main property we want to test.

Property 4 (G6, G7, G8, R3). Whenever RP_j completes a run of the authentication protocol, apparently with a registered authenticator belonging to user *uid* (and whose public key is in L_j), then an authenticator belonging to user *uid* has previously been running the protocol, apparently with RP_j, and the two agents agreed on the data values corresponding to all the variables in p, and each such run of the protocol on RP_j corresponds to a unique run of the protocol on the authenticator:

$$\begin{aligned}&\forall uid, i, L_j, RP_j, p, t_1, t_2, t_4.\\ &\quad \mathsf{A\_Register}(uid, i, x_i, L_j, RP_j)@t_1 \wedge\\ &\quad \mathsf{R\_Register}(RP_j, L_j)@t_2 \wedge\\ &\quad \mathsf{R\_Commit}(RP_j, L_j, p)@t_4 \;\Rightarrow\\ &\qquad \big(\exists i', x_{i'}, t_3.\ \mathsf{A\_Running}(uid, i', x_{i'}, L_j, RP_j, p)@t_3 \wedge\\ &\qquad\quad (t_3 < t_4)\big) \wedge\\ &\qquad \neg\big(\exists t'_4.\ \mathsf{R\_Commit}(RP_j, L_j, p)@t'_4 \wedge \neg(t_4 = t'_4)\big).\end{aligned}$$

Events A_Register and R_Register indicate a successful registration of the authenticators belonging to user *uid*. Event R_Commit indicates a successful authentication at the RP side. If all three events are found in a trace, it must be the case that a legitimate authenticator executed A_Running before the RP executed R_Commit. The last line in the formula above signifies the injectivity between the events A_Running and R_Commit, i.e., each successful execution must correspond to a distinct run of the authentication protocol.

6.3 Macro Language

To accommodate an arbitrary (but finite) number of authenticators, we have developed an extension to the ProVerif syntax that expands a block of code into a specified number of repetitions, in our case the number of authenticators n per user. The extension syntax is:

```
{⟨prefix⟩$⟨var_name⟩;⟨suffix⟩/⟨separator⟩}
```

where `⟨prefix⟩` and `⟨suffix⟩` are constant strings, `⟨var_name⟩` is the name of the variable to be substituted, and `⟨separator⟩` is the string inserted between substitutions (defaulting to "`, `" if unspecified). This syntax allows to parameterize ProVerif constructs.

For example, a list L of an arbitrary number of public keys

```
let PkList({y$i;}) = L in
```

will expand for three authenticators ($1 \le i \le 3$) as

```
let PkList(y1, y2, y3) = L in
```

by calling the macro preprocessor with the appropriate range for the parameter i. The syntax supports nested constructs (within the `⟨prefix⟩` or the `⟨suffix⟩`) using the same or a different variable name. The preprocessor will first expand the innermost blocks of code before proceeding to the outer ones.

7 DISCUSSION

7.1 Generating the Shared Secret

The shared secret k is generated during the introduction stage, during which we assume authenticators A_1 and A_2 are in the same physical location and can communicate securely. The secret k should be known by all authenticators in the set and no other entities.

When there are two authenticators only, the shared secret k can be derived using Elliptic-curve Diffie–Hellman (ECDH) or a similar approach [9]. When there are more than two authenticators, more elaborate algorithms can be used to generate a group key [14]. Alternatively, the secret can be generated by a subset of the authenticators then encrypted and sent to the others. When a new authenticator joins the set, there is no need to establish a new k. The new authenticator can be sent the existing key k by one of the existing authenticators (via some secure channel, such as Bluetooth for example).

7.2 Revocation

Once registered, an authenticator may need to be revoked if the authenticator is lost, broken, or replaced. In all our solutions, the user must begin a session with the relying party, by successfully authenticating, before they can change the list of authenticators. This also applies to existing solutions like FIDO2.

In Solution 1, a single public key is linked with the user's account. The corresponding private key is held by each authenticator. Thus, to revoke one authenticator, the user must revoke all authenticators. The user can then be given the choice to register an updated set of authenticators or to use a different method of authentication.

In Solution 2, multiple public keys are linked with the user's account: each corresponding to a distinct authenticator. Therefore, each authenticator can be revoked independently.

In Solution 3, an ordered list of public keys is linked with the user's account. Each entry in the list corresponds to a distinct authenticator. Thus, each one can be revoked without revoking the others. This can be done by updating the ordered list stored by the relying party. However, to generate valid ring signatures in the future, the list must also be updated in the remaining authenticators.

8 CONCLUSION

In this paper, we present solutions to improve the use of discrete authenticators. The FIDO standards are the key precedent in this space [6, 15]. FIDO provides authentication security without passwords by allowing users to assert their identity using an authenticator. However, authenticators can be lost or destroyed. Therefore, FIDO recommends that users have two authenticators and keep one in a safe place. The problem with this approach is that the second authenticator is generally not available during registration, precisely because it is being kept in a safe place. A natural solution to this problem is to allow one authenticator to register all the other authenticators possessed by the user. This would allow the user to carry a single authenticator whilst others are kept in safe locations as backups. We propose three schemes for how this can be done.

Why three solutions? Because each offers different advantages. *Duplicate* requires the least storage and the storage requirement is independent from the number of authenticators. In our other solutions, the relationship is linear. Also, *Duplicate* offers a high level of privacy; namely, authenticators cannot be differentiated by relying parties. *Ring* offers a similar level of privacy as *Duplicate*. In addition, *Ring* provides increased security by using unique key pairs for each authenticator. *Proxy* also offers unique key pairs for each authenticator. However, it does not offer the same level of privacy as *Duplicate* and *Ring*. The main advantage of *Proxy* is that each authenticator can be registered and unregistered independently. This is not the case in our other solutions. This property is particularly useful when the set of authenticators changes frequently.

ACKNOWLEDGMENTS

EPSRC has provided funding through grant EP/R012598/1 "User-controlled hardware security anchors: evaluation and designs."

REFERENCES

[1] Martín Abadi and Véronique Cortier. 2006. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science* 367, 1-2 (Nov. 2006), 2–32.

[2] Martín Abadi and Cédric Fournet. 2001. Mobile values, new names, and secure communication. In *ACM SIGPLAN-SIGACT Symposium on Principles of programming languages (POPL)*. ACM, London (UK), 104–115.

[3] Martín Abadi and Roger Needham. 1996. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering* 22, 1 (Jan. 1996), 6–15.

[4] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 2002. 1-out-of-n signatures from a variety of keys. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, Queenstown, New Zealand, 415–432.

[5] Fatma Al Maqbali and Chris J Mitchell. 2018. Email-based password recovery-risking or rescuing users?. In *International Carnahan Conference on Security Technology (ICCST)*. IEEE, Montréal, Canada, 1–5.

[6] FIDO Alliance. 2021. Client to Authenticator Protocol (CTAP) Proposed Standard. https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-20210615.html.

[7] FIDO Alliance. 2022. FIDO Security Reference. https://fidoalliance.org/specs/common-specs/fido-security-ref-v2.1-ps-20220523.html.

[8] Myrto Arapinis. 2008. *Security of Cryptographic Protocols: Decidability and Reduction Results*. Ph.D. Dissertation. Paris-East Créteil University.

[9] Elaine Barker, Lily Chen, Sharon Keller, Allen Roginsky, Apostol Vassilev, and Richard Davis. 2017. *Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography*. Technical Report. National Institute of Standards and Technology.

[10] Bruno Blanchet. 2016. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends in Privacy and Security* 1, 1-2 (2016), 1–135.

[11] Bruno Blanchet, Vincent Cheval, and Marc Sylvestre. [n.d.]. ProVerif: Cryptographic protocol verifier in the formal model. http://prosecco.gforge.inria.fr/personal/bblanche/proverif/.

[12] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. 2023. *ProVerif 2.05: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial.*

[13] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. 2012. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *IEEE Symposium on Security and Privacy*. IEEE, San Francisco, CA, 553–567.

[14] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. 2001. Provably authenticated group Diffie-Hellman key exchange—the dynamic case. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, Gold Coast, Australia, 290–309.

[15] World Wide Web Consortium. 2021. Web Authentication: An API for accessing Public Key Credentials Level 2 W3C Recommendation. https://www.w3.org/TR/webauthn/.

[16] Véronique Cortier and Steve Kremer. 2014. Formal Models and Techniques for Analyzing Security Protocols: A Tutorial. *Foundations and Trends in Programming Languages* 1, 3 (2014), 151–267.

[17] Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Transactions on Information Theory* 29, 2 (March 1983), 198–208.

[18] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. 1999. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols*. Trento, Italy.

[19] ISO/IEC. ISO/IEC 11770-6:2016. Information technology – Security techniques – Key management – Part 6: Key derivation.

[20] Juan Lang, Alexei Czeskis, Dirk Balfanz, Marius Schilder, and Sampath Srinivas. 2016. Security keys: Practical cryptographic second factors for the modern web. In *International Conference on Financial Cryptography and Data Security*. Springer, Christ Church, Barbados, 422–440.

[21] Gavin Lowe. 1997. A hierarchy of authentication specifications. In *IEEE Computer Security Foundations Workshop (CSFW)*. IEEE, Rockport, MA, 31–43.

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

[23] Ronald L Rivest, Adi Shamir, and Yael Tauman. 2001. How to leak a secret. In *International conference on the theory and application of cryptology and information security*. Springer, Gold Coast, Australia, 552–565.

[24] Claus-Peter Schnorr. 1991. Efficient signature generation by smart cards. *Journal of cryptology* 4, 3 (1991), 161–174.

[25] Anthony Spadafora. 2021. Struggling with password overload? You're not alone. https://www.techradar.com/uk/news/most-people-have-25-more-passwords-than-at-the-start-of-the-pandemic.

[26] Yongqi Wang, Thalia Laing, José Moreira, and Mark D. Ryan. 2024. Repository of ProVerif models. https://github.com/jmor7690/acm-codaspy2024-authenticators.