Authentication with What You Have: Improving Resilience and Usability

MSc Thesis

Yongqi Wang

UNIVERSITY OF BIRMINGHAM

School of Computer Science
Lead Supervisor: Professor Mark D. Ryan
Co-Supervisor: Dr David Oswald

2022

UNIVERSITY OF
BIRMINGHAM

University of Birmingham Research Archive

e-theses repository

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.

Abstract

This thesis aims to improve the resilience and usability of authentication with what you have. I identify two problems which are pertinent to this area and present solutions to each.

The first problem concerns the remote registration of multiple authenticators. 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 thesis, I present three solutions which combine the usability of a single authenticator with the resilience of multiple authenticators. I also present novel key derivation functions which are important components of the 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 the solutions is analysed using ProVerif.

The second problem concerns secret sharing without a central entity. A cryptographic secret can be shared amongst multiple players such that a subset of players must collaborate to use the secret. A central entity can be tasked to issue new shares of the secret but such an entity must be trusted and secure. This can be costly or impractical. To issue new shares in a decentralised manner, players can take two existing approaches which I call the frontloading approach [20] and the NSG approach [27].

The frontloading approach has the advantages of less communication complexity, fewer requirements to broadcast metadata, more robustness against delays or revoked decisions, and more flexibility in the choice of thresholds.

However, the frontloading approach has limitations which are not yet resolved. These limitations are addressed in this thesis. First, only a limited set of players are able to participate in issuing shares. Second, the setup stage is not fully decentralised. Players need to be provisioned with particular information before they can issue shares and an additional mechanism is required to generate this information in a decentralised manner.

To extend the ability to issue shares, I describe a protocol of multi-layered secret sharing. This includes a protocol for distributing and recombining shares without compromising the security of the shared secret. To make the frontloading approach fully decentralised, I describe a protocol of multi-layered secret sharing with multiple parties. Thus, players can generate the information required for frontloading without a central entity. These solutions may be used to improve the security and usability of secret sharing. This can be valuable for users and service providers in applications such as user authentication and threshold authorisation.

The solutions to the first problem, remote registration of multiple authenticators, are the subject of a paper which is currently in progress. In addition, with colleagues at HP Labs, I have filed two patents on the subject. The solutions to the second problem, secret sharing without a central entity, are the subject of a third patent which I have filed with HP Labs.

Patents and Papers in Progress

1. Yongqi Wang, Thalia Laing, José Moreira, and Mark D. Ryan. Remote registration of multiple authenticators. Paper in progress.

2. Yongqi Wang, Mark D. Ryan, and Thalia Laing. Enabling registration of multiple authentication devices in a challenge/response protocol whilst only one device is present without changes to a FIDO compatible relying party. Patent filed by HP Labs, August 2021.

3. Yongqi Wang, Mark D. Ryan, and Thalia Laing. Enabling registration of multiple authentication devices in a challenge/response protocol whilst only one device is present. Patent filed by HP Labs, June 2021.

4. Yongqi Wang, Thalia Laing, and Josh Schiffman. Issuing shares of a secret without a central entity. Patent filed by HP Labs, August 2020.

Acknowledgments

This thesis is submitted with thanks to Mark Ryan, my supervisor, for his sage guidance and expertise throughout my time at Birmingham and, more recently, whilst I have been working in London; Thalia Laing and Jose Moreira for being talented and thoughtful co-authors on our remote registration paper; David Oswald for being my co-supervisor at Birmingham; HP Labs for their generous funding and the opportunity to work with the excellent team in Bristol; Joshua Schiffman, Simon Shiu, Chris Dalton, and Boris Balacheff who all contributed time to review my work; Nigel Edwards and Ludovic Jacquin for their support and encouragement to undertake this degree; and my family for the countless actions which form the foundation for everything else.

List of Figures

1 Message sequence for Solution 1: Duplicate Authenticators . . . 18
2 Message sequence for Solution 2: Proxy Authenticators 19
3 Message sequence for Solution 3: Ring Authenticators 20
4 An unassigned share of the secret is split into layers of subshares and distributed to two players (A and B) of the first generation. 27
5 Subshares are sent between four generations of players such that player G can compute the unassigned share of the secret. 28
6 Three players (A, B, and C) generate pre-secret keys and split them into layers of subshares. 29
7 Subshares are exchanged amongst the three players. 30
8 Subshares sent by two players are used by the recipient to compute one share of each pre-secret key. Their sum is a distinct share of the secret. 30

List of Tables

1 The trade-off between usability and resilience when using discrete authenticators 10
2 Differences between three remote registration solutions 15
3 Notation for remote registration solutions 22

Contents

Abstract i

Patents and Papers in Progress ii

Acknowledgments iii

List of Figures iv

List of Tables iv

1 Introduction 1

2 Preliminaries 2
- 2.1 Authentication . . . 2
- 2.2 Authentication with What You Have . . . 4
- 2.3 FIDO2 Specifications . . . 5
- 2.4 FIDO2 Functionality . . . 5
- 2.5 Analysis of Security and Privacy . . . 6
- 2.6 Analysis of Usability and Resilience . . . 7

3 Remote Registration of Multiple Authenticators 9
- 3.1 Introduction . . . 9
- 3.2 Preliminaries . . . 11
- 3.3 Design Goals . . . 13
- 3.4 Solutions . . . 14
- 3.5 Key Derivation . . . 17
- 3.6 Discussion . . . 22
- 3.7 Conclusion . . . 23

4 Secret Sharing Without a Central Entity 24
- 4.1 Introduction . . . 24
- 4.2 Background . . . 25
- 4.3 Solutions . . . 26
- 4.4 Conclusion . . . 31

5 Conclusion 32

A Mathematical Description of Secret Sharing 33

B Modelling and Verification of Security Properties for Remote Registration Solutions 37

References 42

1 Introduction

A system of authentication should be sufficiently secure, usable, and resilient. In this context, security is the demarcation of adversaries and legitimate users. Usability is the work required to register and authenticate. Resilience is the ability to maintain functionality when something goes wrong; for example, when an authenticator is lost or broken.

These properties are often in conflict. For example, increasing security by introducing a second authentication factor can decrease usability. As a result, increasing security on paper can have the opposite impact in reality. At some point, users will forgo the secure system in favour of a less secure but more usable alternative. Conversely, improving usability can encourage individuals and organisations to adopt a system which is more secure than the one it replaces. In other words, security can be improved indirectly.

With these properties in mind, some preliminary questions can be asked. What is lacking in the default authentication system of memorised passwords? What are the alternative approaches? What is limiting the adoption of these alternatives?

When using memorised passwords, the lack of security and usability are the main concerns. Users may be able to remember a few strong passwords but not enough for a growing number of accounts. Meanwhile, password cracking tools are sophisticated and relatively easy to use.

Alternative approaches include password managers, two-factor authentication, biometric authentication, and authentication with 'what you have.' In general, these are more secure than memorised passwords. This implies that security is not the limiting factor in their adoption. Thus, improving the security of these approaches is unlikely to be the best way of improving authentication overall. The marginal gains for those who already use these systems are outweighed by the persisting vulnerability of those who do not.

The greatest gains in authentication will likely be found by focusing on users who are not particularly concious of authentication: those with a haphazard collection of relatively weak passwords, reused for various accounts, and linked to an email address protected by a similarly weak password. What would improve authentication for this group?

To a large extent, this can be done without technical novelty. Improvements may be reached through digital education, regulation of data and technology, implementation of secure systems by default, and wider availability of existing hardware such as biometric sensors and security keys. But these initiatives are beyond the scope of this thesis. Instead, the focus will be on novel technical solutions which have a complementary impact.

The aim of this thesis is to improve systems of authentication which are already more secure than passwords but are, nevertheless, underutilized by individuals and organisations. This will be done by identifying the properties which are lacking in existing systems and designing solutions which satisfy these properties. By doing so, more individuals and organisations should be inclined to move away from insecure passwords. Thus, the quality of authentication overall—including security—will be improved.

The importance of usability and resilience, from the perspective of users, is fairly obvious. Whilst security is not noticeable until something goes wrong, usability is noticeable all the time. Meanwhile, losing access to an account—

a matter of resilience—is an obvious concern, more easily imagined than the variety of consequences which can result from unauthorised access.

In summary, the subject of the thesis is the design of authentication systems. The aim is to improve usability and resilience whilst maintaining other properties like security and privacy. The intended result is an improvement in authentication overall, with more people transitioning away from simple password based authentication.

2 Preliminaries

2.1 Authentication

Authentication factors fall into three categories: what you have, what you know, and what you are [18]. 'What you are' consists of biometrics such as fingerprints and face ID. 'What you know' consists of memorised secrets such as a password or PIN. Finally, 'what you have' consists of devices in your possession: this includes general computing devices, such as laptops and smartphones, or special authentication devices such as security keys. Examples include Yubico's *YubiKey* and Google's *Titan* key.

These factors can be deployed individually or as part of a multi-factor system. Passwords, for example, are often implemented as a single factor. In this case, anyone with a password (and username) can access the corresponding account. Biometrics, on the other hand, are usually implemented as part of a multi-factor system. In most cases, they are registered with a particular device. To satisfy the authentication requirements, access to this device is required along with the biometric.

In recent years, there has been widespread acknowledgement that passwords, implemented as a single factor, are insufficiently secure [16]. Moreover, with the proliferation of online accounts, keeping track of an increasing number of passwords is a significant burden. This leads, inevitably, to the use of weak passwords or the reuse of passwords for different accounts.

To strengthen security, a second factor can be required in addition to a password. For example, when a user tries to authenticate on a laptop, they may be asked to input a one-time code along with their password. This code is received on a personal device, most often a smartphone, via an authentication app or a text message. In more sophisticated implementations, rather than manually inputting a code, the user may be asked to swipe their smartphone screen (as proof of presence) or present their fingerprint (as proof of identity). The phone then sends an assertion to the laptop or directly to the relying party.

In the above example, the smartphone is an example of 'what you have' and the fingerprint is an example of 'what you are.' Along with the original password, this creates an authentication system with three 'factors.' Thus, it is reasonable to surmise that security has been strengthened.

However, security is not the only property required in an authentication system. Usability and resilience are other important factors. The effort required to remember passwords is a problem of usability and the likelihood of forgetting passwords is a problem of resilience. The introduction of a second factor does not solve these problems; in fact, it has a negative affect on both. In the example above, the user needs to carry their smartphone with them in order to

authenticate. Moreover, if they lose their smartphone, the system breaks down and they are unable to authenticate.

Another method of improving password based authentication is to use a password manager. In this case, the user creates a single strong yet memorable password—the master password. The user then creates distinct, complex, and random passwords for each of their accounts and puts these in an encrypted vault. These passwords do not have to be remembered. To allow access across different devices, the vault can be stored in the cloud. To authenticate to a particular account, the user first authenticates to the password manager. They might do this by inputting their master password or, when using a previously registered device, by authenticating to their device via a biometric factor. The password manager then provides the user with their encrypted vault.

The security of such a system depends, to a large extent, on the implementation of the password manager and the choice of the master password. For example, the password manager LastPass uses a secure key derivation function, PBKDF2, to derive a vault key and an authentication key from the master password [23]. In this case, the authentication key is sent to the password manager which returns the encrypted vault. The user then decrypts this vault using the vault key in order to access its contents. In theory, the service provider should not be able to compute the user's master password or any of the saved passwords. The same applies to third parties, even if a server storing the encrypted passwords is breached.

Nevertheless, even with a well chosen password and a well implemented password manager, such a system still has vulnerabilities. Most importantly, the master password is a single point of failure which is susceptible to key loggers and phishing attacks. The password manager itself is also a single point of failure: if the vault is corrupted or unavailable, the user is unable to access any of their accounts. In other words, whilst password managers do improve usability by reducing the burden of remembering passwords, they still have limitations in security and resilience.

Another aspect to consider is the difference between perception and technical reality. Even if a system is secure, its adoption can be limited by the difficulty in understanding how the system works and the appearance of insecurity. Although password managers can be secure, the underlying implementation is not obvious to the user. Even if the code is public, most users are not in a position to verify the implementation. They must trust a service which, on the surface, appears as if you are giving a company access to all of your passwords. Unsurprisingly, this might discourage some users from password managers.

In addition to password managers, the inclusion of biometric sensors in more personal devices—laptops as well as smartphones—has significantly improved usability. For users who carry out most of their authentications on personal devices, passwords are required much less frequently. For example, they may only need to input a password (or PIN) when registering a new personal device or accessing an account on a public device. However, whilst this is a marked improvement in usability, it can have a negative impact on resilience: users are more likely to forget their passwords if they are rarely used. Additionally, since the biometric is just an alternative to a password, not a replacement, the security of the system is not significantly improved.

In summary, the use of passwords as a single authentication factor is far from ideal. Problems exist in security, usability, and resilience. Fortunately, signifi-

cant improvements have been made through the development of second-factor authentication, password managers, and biometrics. However, each solution has its limitations. In response, a promising direction of research focuses on authentication with 'what you have.' This will be the subject of the following section.

2.2 Authentication with What You Have

In various systems of authentication, 'what you have' is often involved in a supporting role. For example, after you authenticate to a website using a password, the website may provide a cookie to your personal device. Thus, when using this device—something that you have and others do not—you are able to access your account without re-entering your password. Another example is in biometrics. In general, biometric information is stored on your personal device, not an external server. Thus, access to your personal device is required to facilitate any biometric authentications.

Whilst the notion of 'what you have' has been peripheral to authentication within computer science, it is interesting to note that the possession of an object has been the dominant form of authentication, in other aspects of human civilisation, for thousands of years. Early examples of locks and keys have been found in Iraq which are estimated to be 4000 years old [28]. Meanwhile, 'safe conduct' documents—what would now be called passports—are mentioned in a British Act of Parliament from 1414 [12]. Another example is that of paper banknotes: devised to show that the person in possession of the banknote had deposited a specified amount of metal coinage. These appeared in China in the 7th century [9].

One explanation for why authentication in computer science has focused on other methods of authentication, other than 'what you have,' is that computers are general purpose machines which carry out a wide range of instructions. They are not intended to perform a single operation (unlike a key); they are not bespoke to a particular individual (unlike a passport); and they do not represent other objects of value (unlike a banknote). In addition, unlike the other objects mentioned above, computers are connected to the internet. Thus, they are connected to a world of potential adversaries and constantly vulnerable to a range of remote attacks. It can also be noted that, for much of their history, computers have been shared by multiple individuals.

However, these underlying factors have evolved. Now, it is unremarkable for some individuals to possess multiple personal computing devices. And many devices, despite being quite sophisticated, only have one purpose. In the case of a security key, like the *YubiKey*, this purpose is to facilitate authentication by storing, generating, and managing complex secrets. There has also been a significant trend towards the inclusion of 'hardware anchors' in general purpose devices. This reduces the risk of remote attacks since hardware anchors are isolated from the wider world. Amongst other things, this means that each device can maintain a constant identity and prove this identity through cryptographic operations. Hardware anchors can be dedicated security chips or secure enclaves within the CPU. The former include the *Trusted Platform Module (TPM)*, Google's *Titan* chip, and Apple's *T2*. The latter include Intel's *SGX*, Arm's *TrustZone*, and *Keystone* (based on RISC-V).

With these developments in technology and material conditions, it is possible

to design systems of authentication with 'what you have' in the foreground. It may even be possible to retire passwords entirely. Some promising advances have come from The FIDO (Fast IDentity Online) Alliance and the World Wide Web Consortium (W3C). This will be the subject of the next section.

2.3 FIDO2 Specifications

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 tech companies and adjacent organisations. Some of the main contributors are Google, Microsoft, Mozilla, and Yubico. Meanwhile, W3C is "the main international standards organization for the World Wide Web" (with Tim Berners-Lee being one of the founding members) [33].

The latest set of specifications in this area, published in 2019, is often refereed to as 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. CTAP is published by the FIDO Alliance and WebAuthn is published by W3C.

The primary elements involved in FIDO2 are the authenticators, clients, client devices, and relying parties [18]. 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 [18].

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 both be used as roaming authenticators when accessing accounts on another device. Smartphones and wearables have the advantage of being something which many people already carry with them. Security keys have the advantage of being relatively inexpensive and requiring no battery. They also have a fairly intuitive user experience—analogous to a traditional key—which may be beneficial to certain user groups.

Next, the clients are web browsers including *Chrome*, *Edge*, *Firefox*, and *Safari*. The role of the client is to 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. The latter includes *Android*, *iOS*, *macOS*, and *Windows* [6]. One client device may support multiple clients; for example, both *Firefox* and *Chrome*, with each having the same access to available authenticators [18]. Client devices can range from desktops to laptops, tablets, and smartphones.

Finally, the relying parties are those who provide the web applications which use the Web Authentication API to register and authenticate users [18].

2.4 FIDO2 Functionality

Essentially, FIDO2 is a series of interactions between pairs of elements: user and authenticator, authenticator and client, client and relying party. Compared to

the large number of relying parties and authenticators, there is a much smaller number of clients. Rather than interacting directly, the relying parties and authenticators communicate via the clients. This makes development easier; the authenticators and relying parties only need to be compatible with the clients to be compatible with each other.

The interaction between user and authenticator can take different forms with varying degrees of security. To simply verify user presence, a touch sensor or a button can be used. To guard against attackers with physical access to the authenticator, a PIN or a biometric can be added. In this case, the extra factor just unlocks the authenticator; there is no need to transfer additional data to the relying party.

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 [18]. This is a passwordless challenge response protocol using public key cryptography.

There are two main processes in FIDO2: registration and authentication. During registration, the authenticator creates a public-private key pair which is unique to the relying party. The uniqueness preserves privacy and prevents the linking together of activity across different services. The relying party stores the public key, associated with a particular account, for future authentications. The private key does not necessarily need to be be stored by the authenticator; instead, it can be generated from a root key when required.

The authenticator may also provide an attestation. The minimal requirement for the attestation is determined by the relying party. Some authenticators, such as the *YubiKey*, will come with an attestation key which is embedded during manufacture. The same attestation key is shared amongst many authenticators; this simplifies the verification process and prevents the tracking a particular authenticator by logging the attestation key.

The authentication process uses the same channels of communication as the registration process. Instead of creating a new key pair, the authenticator uses a signature to prove that it is the owner of the private key corresponding to the public key stored by the relying party.

2.5 Analysis of Security and Privacy

Having considered FIDO2 as a concrete example, we can discuss the positives and negatives of authentication with 'what you have' in more detail. In particular, how does it compare with the default approach of memorised passwords?

The security of passwords can be undermined in various ways. To begin with, given a large and growing number of online accounts, it is inevitable that people will use weak passwords or reuse the same passwords across different accounts. Even if relatively strong passwords are used, there are various attacks which apply to each level of the system. For example, phishing, keyloggers, replay attacks, and server breaches.

Over time, a number of solutions have augmented password based authentication. For example, password managers to keep track of strong passwords, federated authentication to reduce the number of passwords overall, sophisticated hashing operations to make it difficult to recover passwords in the event of a server breach, the use of random salts to prevent replay attacks, and the addition of one time passcodes delivered through SMS or an authentication app.

However, various concerns remain. How do people authenticate to their password manager? Is the password used for federated authentication sufficiently secure to protect multiple accounts? How do people know whether a particular service uses strong hash functions and random salts? What if the SMS used to deliver a one time passcode is intercepted? What if an attacker manages to phish the one time passcode along with the password?

In contrast, authentication with 'what you have' can be designed without a password or any other shared secret between the user and the relying parties. Instead, the system can be built on public key cryptography and secure hardware anchors. As is the case in FIDO2 [6]. This prevents phishing and undermines the impact of server breaches. It also prevents credential theft since the user does not need to input a password and the secret data—the private keys and biometric data—never leave the authenticator.

Of course, if there is no PIN or biometric to protect the authenticator, then attackers with physical access can use the authenticator. That said, distinct authenticators, like security keys, usually do not display information like usernames or a list of registered accounts. And other devices, like smartphones, usually have a biometric sensor or keypad which can be used to safeguard the authenticator.

As for privacy, passwords by themselves are not a cause for concern. However, since passwords are easy to forget, almost all relying parties will implement a recovery process which requires an email address. Thus, activity across multiple accounts can be linked by observing the accounts which are linked to the same email address. Meanwhile, more sophisticated password-based solutions may leak some metadata. For example, a password manager can track when a user accesses their encrypted vault.

At first, a system based on 'what you have' seems to present similar challenges to privacy. After all, the same security key or hardware anchor is being used to authenticate to multiple accounts. However, this problem can be overcome by using different public-private key pairs for each relying party. As is the case in FIDO2 [6]. Thus, the privacy of authentication with 'what you have' is arguably better than authentication using passwords.

2.6 Analysis of Usability and Resilience

Whilst solutions like FIDO2 offer significant improvements in security, compared to passwords, the differences in usability and resilience are more subjective. Both approaches present some disadvantages.

In the case of passwords, usability is impacted by the friction of imputing passwords and resilience is impacted when passwords are forgotten. In other words, each authentication attempt requires work from the user and a significant proportion of authentication attempts will fail. For accounts which are infrequently used, the recovery service—resetting a password via email, for example—becomes the de facto method of authentication.

In contrast, when using a solution like FIDO2, the user does not need to recall any passwords. Thus, there should be less friction during input and a lower proportion of authentication attempts which fail. Even when the authenticator is protected by a PIN or biometric, this step should be less burdensome than a traditional password. The PIN does not need to be as complex as a password

since it is only used locally. Also, the registration and authentication process across different services should be simple and consistent.

Moreover, associations between authenticators and accounts are quite flexible. Effectively, an unlimited number of accounts can be registered with each authenticator and multiple authenticators can be registered with each account. Since authenticators can compute authentication keys using a root key and fixed information about the relying party, the number of accounts is not restricted by the memory available to the authenticator.

However, authentication with 'what you have' also comes with significant disadvantages in usability. Most notably, users will need to keep track of their authenticators and may need to carry an additional device. This applies, in particular, when the user's client device does not have an in-built authenticator or when they encounter a new device. More importantly, without an authenticator and a compatible client device, users cannot access their accounts. Although this might happen rarely, the edge cases are often the most inconvenient. For example, if someone loses their phone and security key when travelling and they need to access their bank account.

Resilience is also a problem in FIDO2 and similar systems. If an authenticator is lost or broken, this can entail lengthy recovery processes or the loss of accounts. Recovery processes are not standardised across different services. To avoid this, back up authenticators are recommended. However, this impacts usability since the user will need to perform multiple registration ceremonies and keep track of multiple authenticators. This problem is discussed in more detail in Sec. 3.1.

Finally, the security of a system based on 'what you have' can limit the set of use cases. For example, a password can be easily shared with someone in a different location. This is not currently possible in the case of FIDO2.

The solutions presented in the following chapter offer users the convenience and resilience of registering multiple authenticators remotely. They also make it easier for users to securely grant others access to an account if they wish to.

3 Remote Registration of Multiple Authenticators

The work in this chapter is the subject of a paper currently in progress. It is also the subject of two patents I filed with HP Labs in 2021. The paper and, subsequently, this chapter were written in collaboration with Mark Ryan, Thalia Laing, and Jose Moreira. In particular, Appendix B, showing the modelling and verification of security properties, was written by Jose Moreira. For the other sections, I wrote the initial drafts and other authors may have added lines or rearranged and rewritten certain paragraphs.

3.1 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, 18].

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 [16]. There are, however, significant limitations which impede the adoption of these methods and the move away from passwords [25].

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 [4]. Therefore, users are advised to possess more than one authenticator [22].

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

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 backups in a safe location	Limited	Good

Table 1: The trade-off between usability and resilience when using discrete authenticators

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 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 registration ceremonies. 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. 3.4.

We also present two novel key derivation functions which are fundamental to our three 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. 3.5.

A secondary problem which impacts discrete authenticators, compared to passwords, is the inflexibility of sharing accounts. Whilst a password can easily be shared using encrypted instant messaging, a discrete authenticator cannot.

The solutions presented in Sec. 3.4 also solve this problem.

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. We also present two novel key derivation functions used in our solutions. Finally, we analyse the security of our solutions using ProVerif.

Organisation. In Sec. 3.2, we give a brief introduction to the signature schemes used in our solutions. In Sec. 3.3, we discuss our design goals. Our solutions are defined in Sec. 3.4. Sec. 3.5 sets out our key derivation schemes. We analyse the security of our solutions, using ProVerif, in Appendix B. In Sec. 3.6, we include some discussion points for context. We conclude in Sec. 3.7.

3.2 Preliminaries

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.

Schnorr Signatures

In our first two solutions, presented in Sec. 3.4, we use Schnorr signatures [31], 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 $H : \{0,1\}^* \to \mathbb{Z}_q$. The set $\mathcal{P} = (\mathbb{G}, q, g)$ denotes the public parameters of the scheme. Note that, in the following description, we define the function symbol $\mathsf{Pk}(x) = g^x \pmod q$. We also include modular reduction notation. In following sections, we will omit this notation 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:

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

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

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

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

$\mathsf{Verify}(y, m, \sigma) = v$ 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:

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

Ring Signatures

Our third solution uses ring signatures which were formalised in 2001 by Rivest et al. [30]. 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 private key was used.

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 [3]. 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. 3.5.

Abe et al.'s ring signature [3] 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:

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

3.3 Design Goals

The solutions we present in Sec. 3.4 satisfy the following design goals. To overcome the problem described in Sec. 3.1, without requiring a significant increase in resources, we have the following goals.

1. **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.

2. **Convenient storage of back up authenticators.** Apart from an introduction phase, discussed in Sec. 3.4, 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.

3. **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.

4. **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.

5. **No additional entities.** Compared to the FIDO2 Standard [6, 18], no additional entities are required as intermediaries or trusted third parties.

In addition, we include a number of properties from FIDO2 which are important to maintain [5]. Note, FIDO2 has other properties which are beyond the scope of this paper. For our purposes, the key properties are as follows.

6. **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.
7. **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.
8. **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.
9. **Privacy across RPs.** A user's interactions with an RP cannot be linked to their interactions with any other RP.

3.4 Solutions

In this section, we present three solutions to the problem introduced in Sec. 3.1. We call these **Duplicate**, **Proxy**, and **Ring** Authenticators. All three solutions satisfy the design goals, presented in Sec. 3.3, and each relies on a key derivation algorithm, presented in Sec. 3.5. Differences between the solutions, which are advantageous in different scenarios, are outlined in Table 2.

Duplicate Authenticators require the least storage. In the Proxy and Ring solutions, the number of public keys stored, and the amount of memory required, increases linearly with the number of authenticators. In the Duplicate solution, each authenticator is only required to store one public key.

Minimising memory is valuable when authenticators are embedded in everyday objects. In addition to being secure and usable, these implementations need to be small, simple, and inexpensive.

Duplicate and Ring Authenticators both offer a high level of privacy. Valid signatures from different authenticators, authenticating to the same account, cannot be distinguished by the RP. This prevents advertisers and other entities from tracking patterns of authenticator use. This data could be used to infer the location of a user and the devices they are using.

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 valued over memory requirements.

Finally, Proxy Authenticators are particularly beneficial when the set of authenticators often changes. Each authenticator can be registered and revoked

Property	Duplicate	Proxy	Ring
The authenticators can be registered and revoked independently (see Sec. 3.6).	✗	✓	✗
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.	✓	✗	✗

Table 2: Differences between three remote registration solutions

easily and independently. This is because the public keys are entirely independent, which is not the case with Duplicate or Ring Authenticators.

In addition, similar to Ring Authenticators, every authenticator has its own unique key. Therefore, Proxy Authenticators offer a convenient and 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 various everyday authentication scenarios.

These solutions also solve a secondary problem; namely, the problem of granting another user access to an account remotely. In the case of Duplicate Authenticators, anyone who has access to one authenticator in the set is able to access all the accounts to which the authenticators are registered. In the case of Ring Authenticators, a user is able to grant someone else access to an account by adding the appropriate public key to the list L_j for relying party RP_j. This list would also need to be updated across the other authenticators in the set.

In the case of Proxy Authenticators, a user is able to grant someone else access to an account by registering the public key of an authenticator belonging to the other user. Access can be granted and revoked without physical interaction (unlike the Duplicate solution) and without updating the other authenticators in the set (unlike the Ring solution). Note, an additional mechanism is required to distinguish the original user and the second user. Otherwise, the second user may revoke the authenticator of the original user.

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 secret k. It is repeated only when a new authenticator is added to the set.

The registration stage requires 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.

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 registration and authentication 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. 3.5. Sign and Verify are the signature and verification algorithms defined in Sec. 3.2.

$\mathsf{Verify}(y_{ij}, m_i, \sigma_{ij})$, in Solution 1, requires the following computation:

$$\begin{aligned}\hat{c} &= H(y_{ij} \parallel g^s\,(y_{ij})^c \parallel m_i)\\ &= H(y_{ij} \parallel g^{r-cx_{ij}}\,(y_{ij})^c \parallel m_i)\\ &= H(y_{ij} \parallel g^r\,(y_{ij})^{-c}\,(y_{ij})^c \parallel m_i)\\ &= H(y_{ij} \parallel g^r \parallel m_i) \quad = \quad c.\end{aligned}$$

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. 3.5, 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. Sign and Verify are the signature and verification algorithms defined in Sec. 3.2.

$\mathsf{Verify}(y_{ij}, m_i, \sigma_{ij})$, in Solution 2, requires the following computation:

$$\begin{aligned}\hat{c} &= H(y_{ij} \parallel g^s\,(y_{ij})^c \parallel m_i)\\ &= H(y_{ij} \parallel g^{r-c(x_i k_{ij})}\,(y_i^{k_{ij}})^c \parallel m_i)\\ &= H(y_{ij} \parallel g^r\,(y_i)^{-ck_{ij}}\,(y_i)^{ck_{ij}} \parallel m_i)\\ &= H(y_{ij} \parallel g^r \parallel m_i) \quad = \quad c.\end{aligned}$$

Solution 3: Ring Authenticators

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

$\mathsf{RingSign}\big((x_{2j}, L_j, (\text{AUTH}, m_2)\big)$. The protocols for this solution are depicted in Fig. 3. RingSign and RingVerify are the ring signature and verification algorithms defined in Sec. 3.2.

$\mathsf{RingVerify}\big((y_{1j}, y_{2j}), m_i, \sigma_{ij}\big)$, in Solution 3, requires the following computation (assuming the signature was created by A_1):

$$\begin{aligned}
\hat{c} &= H\big((y_{1j}, y_{2j}) \parallel g^s\,(y_{1j})^{c_1}\,(y_{2j})^{c_2} \parallel m_i\big) \\
&= H\big((y_{1j}, y_{2j}) \parallel g^{r-c_1(x_1 k_{1j})}\,(y_1^{k_{1j}})^{c_1}\,(y_{2j})^{c_2} \parallel m_i\big) \\
&= H\big((y_{1j}, y_{2j}) \parallel g^{r}\,(y_1)^{-c_1 k_{1j}}\,(y_1)^{c_1 k_{1j}}\,(y_{2j})^{c_2} \parallel m_i\big) \\
&= H\big((y_{1j}, y_{2j}) \parallel g^{r}\,(y_{2j})^{c_2} \parallel m_i\big) \;=\; c \;=\; c_1 + c_2.
\end{aligned}$$

Finally, the three solutions discussed require the common security practice of tagging the registration and the authentication protocols. 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. Recall that the security of a tagged protocol does not imply the security of its untagged version.

3.5 Key Derivation

In this section, we present two novel key derivation algorithms, Der and DerMulti, which augment the standard key generation algorithms outlined in Sec. 3.2 and 3.2. 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 unique key pair per RP *and* per authenticator by using both the shared secret k, and a unique 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. 3.3.

R2. **Repeatable key derivation.** Key pairs can be derived deterministically and on demand. This minimises the amount of 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 the 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.

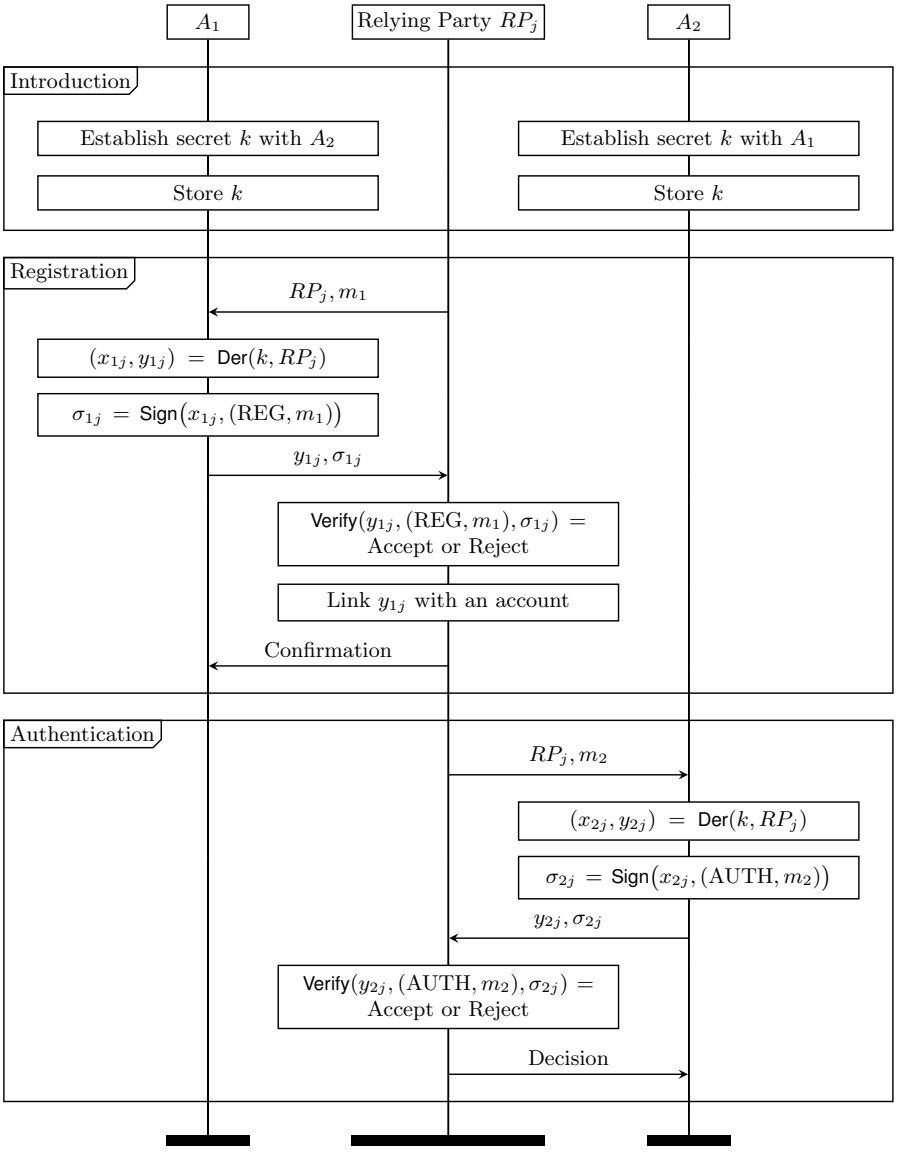

Figure 1: Message sequence for Solution 1: Duplicate Authenticators

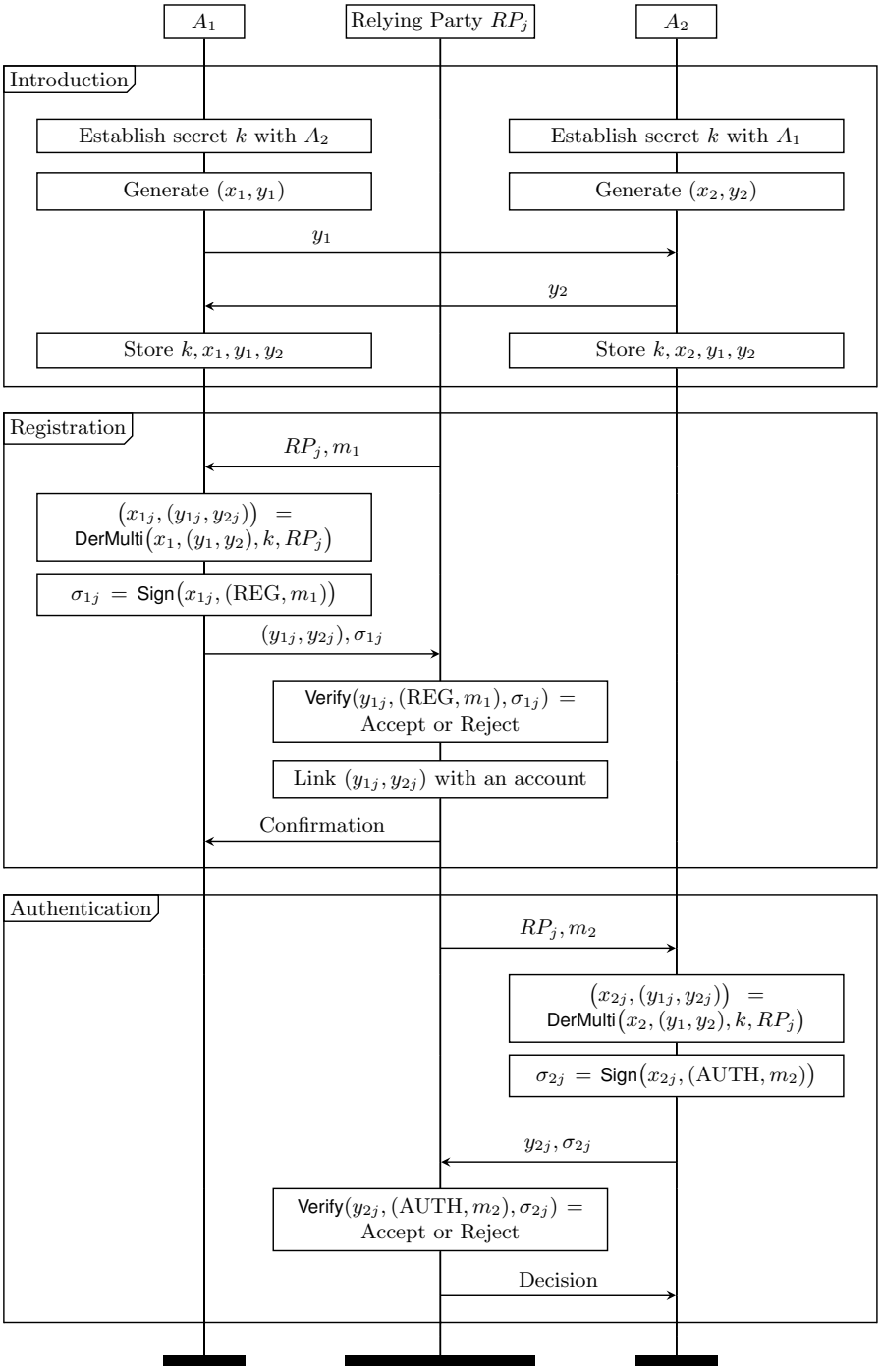

Figure 2: Message sequence for Solution 2: Proxy Authenticators

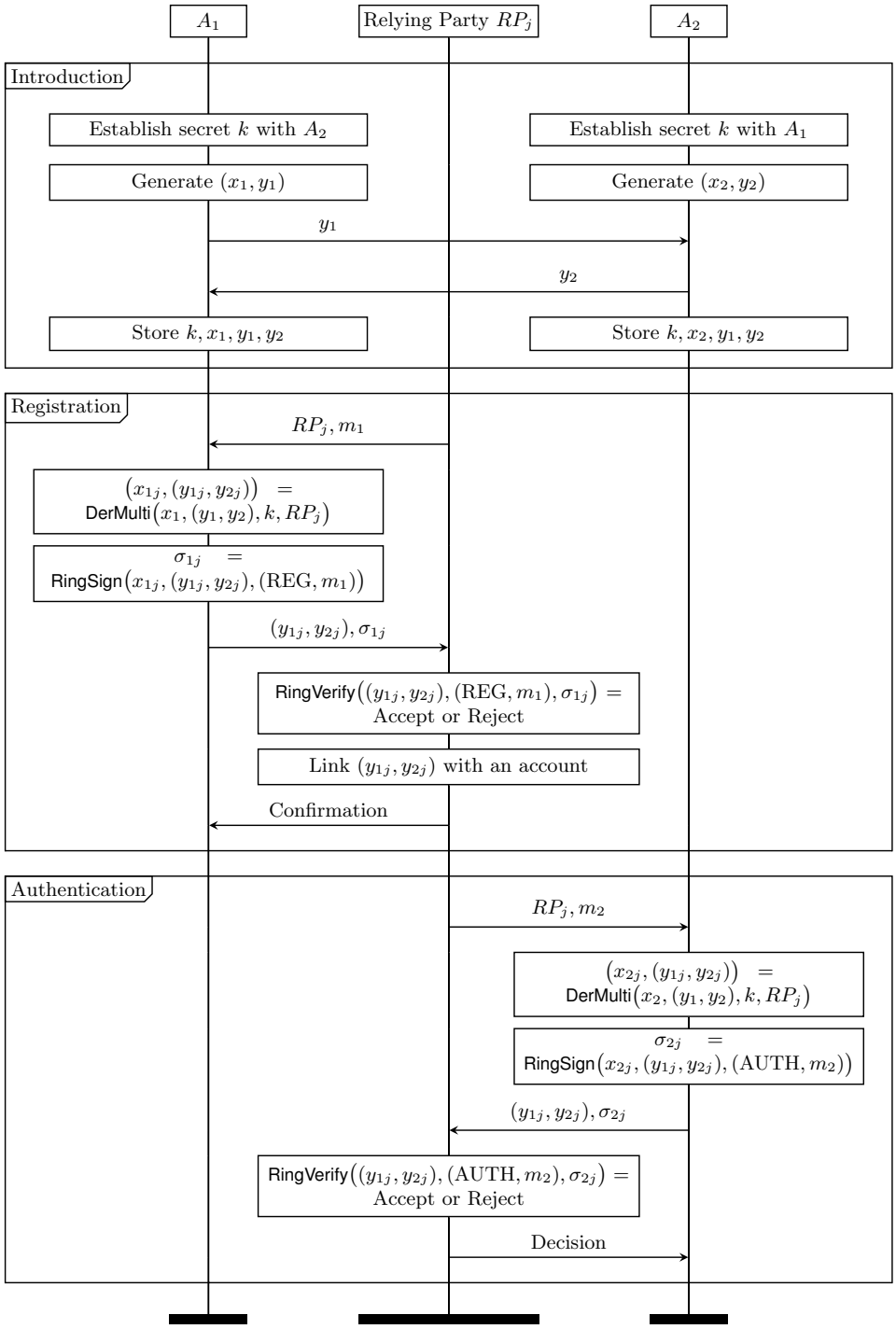

Figure 3: Message sequence for Solution 3: Ring Authenticators

R6. **Compatibility with chosen signature schemes.** Finally, the derived key pairs should be compatible with the appropriate signature scheme for each solution in Sec. 3.4.

The two key derivation algorithms executed by the authenticators rely on a shared secret k. Let us assume that 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}.

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

$\mathsf{DerMulti}(x_i, L, k, RP_j) = (x_{ij}, L_j)$ is a key derivation algorithm, which takes as input a private key x_i, a list of public keys $L = (y_1, \ldots, y_n)$, a shared secret k, and an RP identifier RP_j, and outputs a private key x_{ij} and a list of public keys $L_j = (y_{1j}, \ldots y_{nj})$. 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:

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

2. Compute other authenticator's public keys:

$$\text{for } 1 \leq i' \leq n:$$
$$k_{i'j} := H(y_{i'} \parallel k \parallel RP_j)$$
$$y_{i'j} := \mathsf{DerPk}(y_{i'}, k, RP_j) \stackrel{\text{def}}{=} y_{i'}^{k_{i'j}}.$$

We can now confirm that Der and DerMulti satisfy the requirements above. R1 follows from the uniqueness of RP_j and 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 does not reveal any information about the pivate key. For DerMulti, R3 depends on the fact that only authenticators in the set can compute k_{ij}, as only they know k, and each private key, x_i, is only known to one authenticator. Thus, the output of the multiplication $x_i \cdot k_{ij}$ 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 are unable to compute y_{ij} without knowledge of k_{ij}. Finally, R6 follows by definition.

Symbol	Description
A_i	Authenticator $i \in \{1, \ldots, n\}$.
RP_j	Unique identifier of relying party j.
k	A strong secret shared by all the authenticators.
L	A list of public keys, as defined in Sec. 3.2.
L_j	A list of public keys associated with RP_j, as defined in Sec. 3.5.
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.

Table 3: Notation for remote registration solutions

Note, 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, i'. On the other hand, DerMulti requires each authenticator to have a unique primary key pair (x_i, y_i) that must be generated in advance. 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 is able to obtain its own secondary key pair (x_{ij}, y_{ij}) and the secondary public keys $y_{i'j}$ of the other authenticators for the relying party RP_j.

For all x_i, k, RP_j,

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

This is the algebraic relation that enables any authenticator to derive the public keys of the other authenticators.

Note, 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 $y_{1j}, \ldots, y_{nj}$ associated with relying party 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.

3.6 Discussion

Generating the Shared Secret

The shared secret k is generated during the introduction stage. At this stage, we assume the authenticators A_1 and A_2 are in the same physical location and can communicate securely. The resulting secret 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 [11]. When there are more than two authenticators, more elaborate algorithms can be used to generate a group key [17]. Alternatively, the secret can be generated by a subset of the authenticators then encrypted and sent to the others.

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. Therefore, 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. Therefore, each authenticator 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 ordered list must also be updated in the remaining authenticators.

3.7 Conclusion

In this chapter, we present solutions to improve the use of discrete authenticators. The FIDO standards are the key precedent in this space [6, 18]. 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 solutions for how this can be done.

Each of our solutions offers different advantages. *Duplicate* requires the least storage. 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 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.

4 Secret Sharing Without a Central Entity

The work in this chapter is the subject of a patent I filed with HP Labs in 2020.

4.1 Introduction

Secret sharing is increasingly used in cryptographic protocols to improve security, availability, and reliability by spreading shares of a cryptographic secret among multiple parties. Often, a centralised entity, called the dealer, generates the secret and issues the shares. The dealer sees the secret in its entirety; thus, the dealer must be secure against compromise and trusted to handle the secret correctly. However, it can be costly to create and maintain such an entity; the players may be unwilling to trust a central entity; and the dealer is a single point of failure which, if compromised, can impact the security and functionality of the entire system.

A response to these concerns is to limit the scope of the dealer. In some implementations, a dealer is used to generate the secret but has limited availability thereafter. For example, to reduce the attack surface, the dealer can be a device which is kept in a safe location and requires physical access. Alternatively, the dealer can be ephemeral: all traces of the secrets distributed are deleted after the fact. In other implementations, there is no dealer in the first place; instead, the players generate a shared secret in a decentralised manner.

These implementations can limit the need to trust a central entity, the cost of creating or maintaining such an entity, and the risk of a single point of failure. However, they can also limit the ability to issue new shares of a secret. This ability is useful when players are added or when shares must be replaced. For example, consider the application of secret sharing in multi-device based authentication (MDBA): the subject of a previous HP Labs disclosure [10]. Shares of a signing key are distributed amongst a user's devices. The devices then then contribute collectively in an authentication protocol. In this case, it can be necessary to issue a new share if a device is upgraded, lost, or damaged; if a share of the secret is lost or damaged; or if the user simply wants to add another device to their authentication system.

In another HP Labs disclosure [20], the frontloading approach is described for issuing new shares of a secret. Each player is provisioned with information which can be used to issue new shares. To issue a share, a subset of players is required to participate. Each participating player sends information directly to the recipient of the share. From the information received, the recipient can compute a distinct share.

At present, frontloading has the following limitations. These are addressed in this chapter.

1. **Only limited players are able to participate in issuing shares.**

   Currently, players provisioned with information during setup can participate in issuing new shares but players who join later cannot. This limits the capacity of the frontloading approach. In MDBA, for example, new devices cannot be added once the initial devices have retired. In this chapter, we extend the frontloading approach so that players who join after the setup stage can also participate in issuing shares.

2. **An additional mechanism is required to make the frontloading approach fully decentralised.**

   In the frontloading approach, once players are provisioned with the relevant information, they can issue new shares without a central entity. However, players lack a decentralised mechanism for generating this information in the first place. This information cannot be generated by each player individually; instead, the players need to collaborate—without compromising the security of the shared secret. In this chapter, we provide a mechanism to achieve this result.

4.2 Background

Nojoumian, Stinson, and Grainger proposed a scheme which we refer to as NSG [27]. It works as follows. A subset of players agree, as a group, to issue a share. Each player uses their existing share to generate fragments of a new share—using the appropriate Lagrange interpolation coefficient—and sends one distinct fragment to each of the other participating players. Each player sums the fragments in their possession and sends the result to the recipient of the share. The recipient sums everything they receive which gives them a distinct share of the secret.

The NSG approach is mathematically elegant but has disadvantages in the following scenarios.

1. **When the players issuing the share are not connected by secure channels or when communication is costly.**

   The NSG approach requires secure two-way communication between each pair of players issuing the share and secure one-way communication between each of these players and the recipient of the share. If the number of players issuing the share is t, the total number of secure channels required is t^2. In a refined form, the communication complexity can be reduced by around half but the relationship between the number of players and the number of secure channels is still second order [21]. Moreover, the players need to agree as a group to issue a share. This can require additional communication.

   By comparison, frontloading does not require secure communication between the players issuing the share. The players can send information directly to the recipient of the share. Additionally, players receive the information they need to perform frontloading via the same channels used to issue their own shares. Thus, if the number of players issuing the share is t, the number of additional secure channels is also t. Moreover, the players do not need to agree as a group to issue a share.

2. **When the players issuing the share are unable, or unwilling, to broadcast their decisions.**

   When using the NSG approach to issue a share, if a player decides to participate, they must broadcast this decision to multiple players who are not the recipient of the share. Some devices may not have the capacity to make this broadcast. Or, when the devices are linked to people, these people may prefer to keep their decision private.

By comparison, frontloading does not require players to broadcast their decision to participate, or not participate, in issuing a share. Each player can send information directly to the recipient of the share and, if sufficient players participate, the recipient can compute a distinct share. Equally, if a player requests information from others in order to obtain a share, they can make independent requests to individuals rather than making a request to a group.

3. **When the process of issuing shares needs to be robust against delays and revoked decisions.**

   The NSG approach can be disrupted when a player delays their communications or changes their decision. Such a player cannot be replaced without invalidating previous communications. By comparison, these events are less disruptive when using the frontloading approach. In the latter, players can be replaced without invalidating previous communications.

4. **When different thresholds are required for issuing a share and using the secret.**

   When using the NSG approach, the threshold for issuing a share is necessarily equal to the threshold for using the secret. By comparison, when using the frontloading approach, the two thresholds are mathematically independent. The only constraints are security constraints which will vary depending on the context in which secret sharing is deployed.

In summary, compared with the NSG approach, frontloading achieves less communication complexity, fewer requirements for the players to broadcast metadata, more robustness against delays or revoked decisions, and more flexibility in the choice of thresholds for using the secret or issuing a share. However, frontloading in its current form still has substantial shortcomings. First, only a limited set of players are able to participate in issuing shares. Second, the approach is not fully decentralised. Players need to be provisioned with particular information before they can issue new shares; an additional mechanism is required to generate this information in a decentralised manner. The solution presented in this chapter resolves both of these issues.

4.3 Solutions

This solution addresses the aforementioned limitations in the frontloading approach. First, we expand the limited set of players able to participate in issuing shares. Second, we describe a mechanism with which players can generate the information required for frontloading in a decentralised manner.

Extending the ability to issue shares

Currently, only players present during setup can participate in issuing new shares. We now describe a solution in which players who join after the setup stage can also participate. The following is a high-level description of the solution. A mathematical description is included in Appendix A.

1. Define multiple layers of secret sharing starting with the secret itself. Split the secret into shares. Split these shares into subshares. Split these subshares into further subshares, and so on. Each share of the secret is indexed. An unlimited number of shares can be defined at each layer.

2. Distribute shares and subshares amongst the first generation of players. Each player receives a distinct share of the secret and a distinct set of subshares from each layer of secret sharing. Players can receive more than one share of the secret if required.

3. To issue a new share, a sufficient number of players are required to participate. Each participating player sends a subset of the subshares in their possession. The subset is determined by the index of the share being issued.

4. From the subshares received, the recipient can compute a distinct share of the secret. Moreover, they can compute their own set of subshares. Using this set of subshares, they can participate in issuing shares to others.

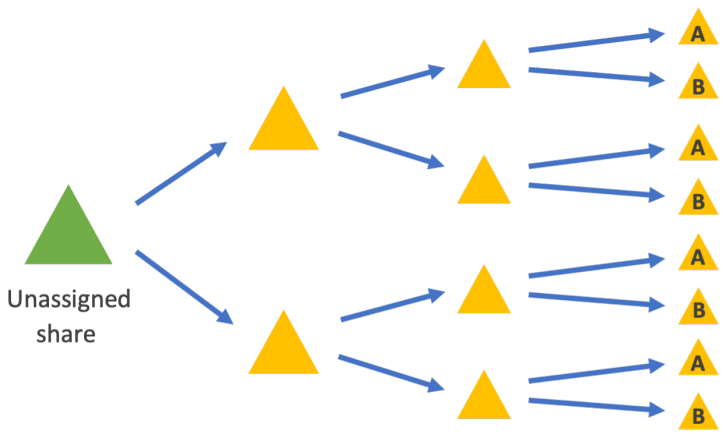

Figure 4: An unassigned share of the secret is split into layers of subshares and distributed to two players (A and B) of the first generation.

Generating information in a decentralised manner

In the frontloading approach, each player is provisioned with information that can be used to issue new shares. In this section, we provide a mechanism for generating this information in a decentralised manner. We build on an existing mechanism, by Pedersen, for generating a shared secret in a decentralised manner [29]. The basic idea in Pedersen's mechanism is that each player generates a pre-secret key and distributes shares of this key to the others. The following is a high-level description of the solution. A mathematical description is included in Appendix A.

1. Each player generates a pre-secret key or psk.

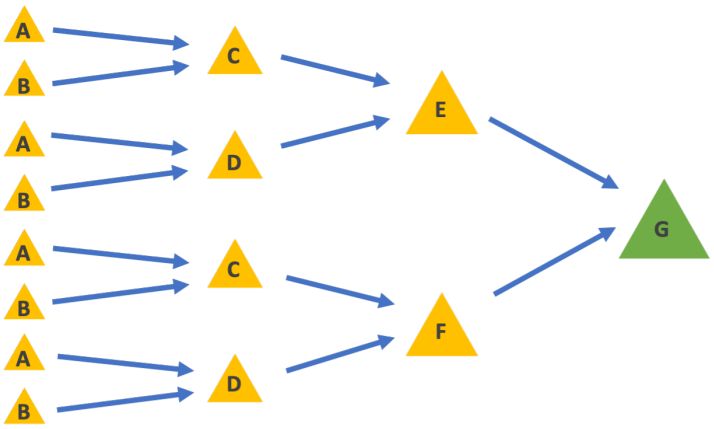

Figure 5: Subshares are sent between four generations of players such that player G can compute the unassigned share of the secret.

2. Each player defines multiple layers of secret sharing starting with their *psk*. Each *psk* is split into shares. Each share is split into subshares, and so on. Each share of each *psk* is indexed. An unlimited number of shares can be defined at each layer.

3. The shares and subshares are exchanged amongst the first generation of players. For each *psk*, the players receive a distinct share and a distinct set of subshares from each layer of secret sharing. Players can receive more than one share of each *psk* if required.

4. The shared secret is defined as the sum of the *psk*s. Each share of the secret is the sum of one share of each *psk*. Each share of the secret is indexed according to the index of the values in the corresponding sum.

5. To issue a new share, a sufficient number of players are required to participate. Each participating player sends a subset of the subshares in their possession. The subset is determined by the index of the share being issued.

6. From the subshares received, the recipient can compute a distinct share of the secret. Moreover, they can compute their own set of subshares. Using this set of subshares, they can participate in issuing shares to others.

Note that the mechanism above can be also run on existing pre-secret keys instead of generating new pre-secret keys. In other words, players who control the pre-secret keys can generate the information required to issue new shares whenever they like. They do not need to do this when the shared secret is first established.

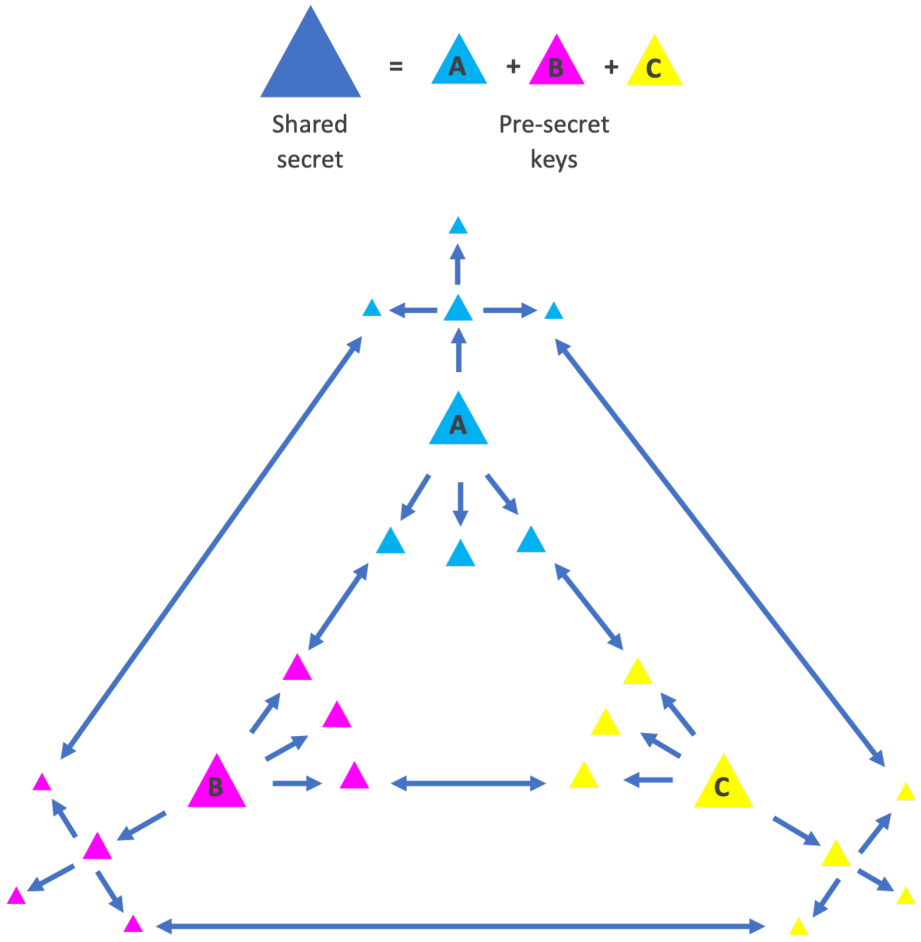

Figure 6: Three players (A, B, and C) generate pre-secret keys and split them into layers of subshares.

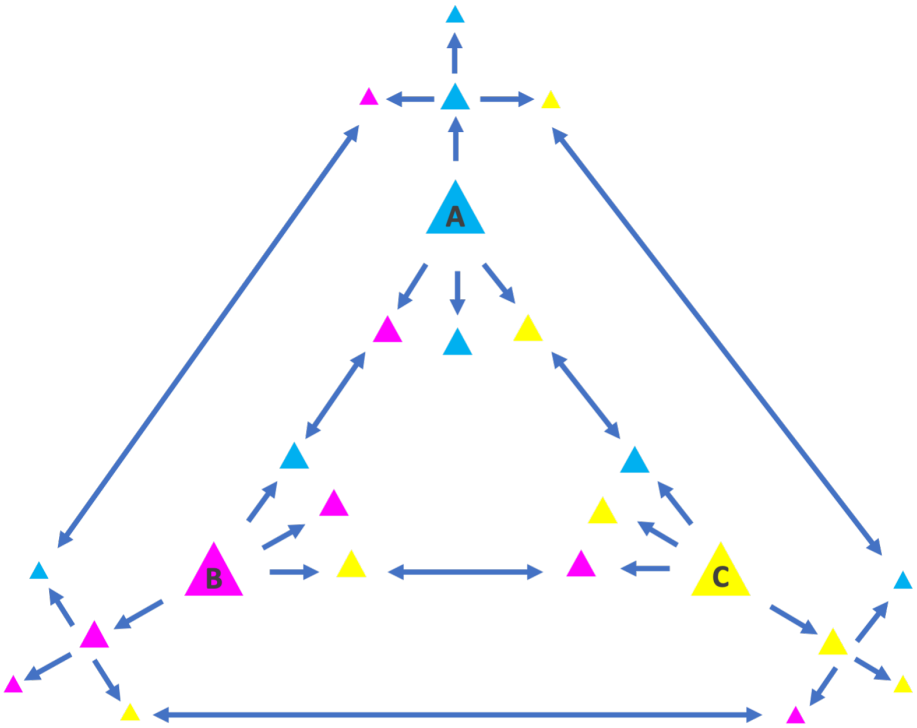

Figure 7: Subshares are exchanged amongst the three players.

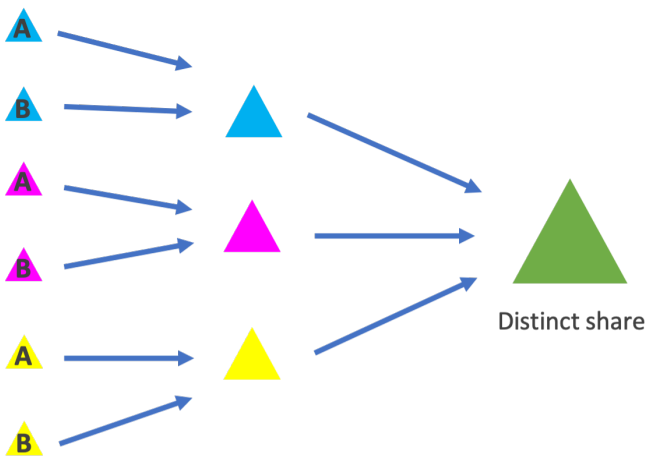

Figure 8: Subshares sent by two players are used by the recipient to compute one share of each pre-secret key. Their sum is a distinct share of the secret.

4.4 Conclusion

Secret sharing can improve security and usability in various applications including user authentication and threshold authorisation.

The ability to issue new shares of a secret without relying on a central entity is useful in the MDBA setting and other settings of secret sharing. A new share can be necessary if a device is upgraded, lost, or damaged; if a share of the secret is lost or damaged; or if the user simply wants to add another device to their authentication system. Meanwhile, for reasons of security and cost, a central entity is not always available to issue new shares. In some implementations of secret sharing, there is no central entity in the first place. Moreover, a centralised entity represents a risk that must be managed by the service provider. Therefore, a solution that removes this risk is valuable.

Various properties are useful in a mechanism for issuing shares without a central entity. This includes low communication complexity, few requirements for the players to broadcast metadata, robustness against delays and revoked decisions, and flexibility in the choice of thresholds. Other important properties include the option of decentralised setup and minimal restrictions on the players who can participate in issuing shares.

The problem of issuing shares without a central entity has been considered in academia and industry, including a recent disclosure from HP Labs [20]. In previous solutions, the properties listed above could only be partially obtained: various properties were incompatible. In this chapter, we provide a solution in which all these properties are obtained. This may be used to improve user authentication, threshold authorisation, and other applications of secret sharing.

5 Conclusion

In this thesis, I have focused on two problems which are relevant to the resilience and usability of authentication with what you have. In particular, I have considered the remote registration of multiple authenticators and secret sharing without a central entity. In addition to this thesis, the solutions I developed are the subject of a paper currently in progress and three patents I filed with HP Labs in 2020 and 2021.

For the remote registration of multiple authenticators, I have presented three solutions which allow users to register all of their authenticators when only one authenticator is physically present. This improves the usability and resilience of authentication schemes, like FIDO2, which rely on discrete authenticators and challenge-response protocols.

Each solution offers different advantages which are discussed in Sec. 3.4. Moreover, each solution satisfies a number of overarching design goals. These include convenient registration and storage of back up authenticators, no significant increase in authenticator or RP requirements, and no additional entities compared to FIDO2. Meanwhile, the solutions maintain core properties from FIDO2 including secure registration and authentication, interchangeable authenticators, and privacy across RPs.

For secret sharing without a central entity, I have presented a solution which satisfies a number of useful properties. Some existing solutions satisfy a subset of these properties but none satisfies all of them.

The properties include low communication complexity, fewer requirements for the players to broadcast metadata, robustness against delays and revoked decisions, flexibility in the choice of thresholds, the option of decentralised setup, and fewer restrictions on the players who can participate in issuing shares. This solution may be used to improve user authentication and other applications of secret sharing.

A Mathematical Description of Secret Sharing

Here we provide some mathematical implementations of the various components in Sec. 4.3. This shows that the scheme is mathematically feasible.

Frontloading in its current form

In the following, t players must collaborate to use the shared secret and t' players must collaborate to issue a new share. Each player has one share of the secret but this can be increased if desired. There are n initial players.

Initial setup

1. The dealer generates a shared secret (referred to as the secret key and denoted SK).
2. For $1 \leq i \leq t-1$, the dealer randomly chooses α_i from $\{0, \ldots, m-1\}$, where m is a previously fixed modulus, to define the polynomial
$$f(x) = \sum_{i=1}^{t-1} \alpha_i x^i + SK.$$
3. For $1 \leq i \leq n$, the dealer computes $s_i = f(i) \mod m$. These are the shares distributed to the n initial players, where s_i goes to player i.
4. For $n+1 \leq j \leq u$, the dealer also computes $s_j = f(j) \mod m$. These are the shares which players can be given in the future.
5. For each $n+1 \leq j \leq u$, and for $1 \leq i \leq t'-1$, the dealer randomly chooses β_i from $\{0, \ldots, m-1\}$ to define a polynomial
$$g_j(x) = \sum_{i=1}^{t'-1} \beta_i x^i + f(j).$$
6. For $1 \leq i \leq n$ and $n+1 \leq j \leq u$, the dealer computes $s'_{ij} = g_j(i) \mod m$. These are the subshares distributed to the n initial players, where $\{s'_{ij} : n+1 \leq j \leq u\}$ goes to player i.

Issuing a share

To issue a share, t' players select a previously unassigned j and send, through a secure channel, their subshares s'_{ij} to the recipient of the share. The recipient uses these subshares to calculate the new share $s_j = f(j) \mod m$. This calculation is done in the same manner as Shamir's scheme [32].

Extending the ability to issue shares

We introduce a third layer of secret sharing—where the subshares s'_{ij} are themselves split into shares. Thus, with n initial players, player i would receive the following:

1. s_i, a share of the shared secret SK
2. $\{s'_{ij} : n+1 \leq j \leq u\}$, a set of shares of the shares $\{s_j : n+1 \leq j \leq u\}$
3. $\{s*_{ijk} : n+1 \leq j \leq u \text{ and } n+1 \leq k \leq u\}$, a set of shares of the subshares $\{s'_{ij} : n+1 \leq j \leq u\}$

When a new player joins the scheme, they are given the subshares $\{s'_{ij}\}$ for some fixed j along with the shares of subshares $\{s*_{ijk} : j < k \leq u\}$. With these two sets, the player can compute a distinct share of the secret key SK along with a set of subshares which can be used, with the help of $t'-1$ other players, to issue further shares.

In the same manner as above, more layers of secret sharing can be introduced. Thus, we achieve multiple layers of 'nested' secrets. We can define any number of layers and, for each polynomial $f(x)$ used in any layer, any number of points $(x, f(x))$ to act as shares for the secret $f(0)$.

Generating a shared secret in a decentralised manner

In the following, t players must collaborate to use the shared secret [29]. Each player has one share of the secret but this can be increased if desired. There are n players in total.

1. Each player generates a pre-secret key, denoted psk_i and a polynomial of the form

$$P_i(x) = \sum_{j=0}^{t-1} \alpha_{i,j} x^j$$

   where $\alpha_{i,0} = psk_i$ and all other coefficients are random.

2. Each player computes $n-1$ shares of their psk, $P_i(1), \ldots, P_i(n-1)$, and distributes one share to each of the other players.

3. Each player computes their share of the secret where $F(j) = \sum_{i=1}^{n} P_i(j)$ is the share of player j.

The shared secret is $SK = psk_1 + psk_2 + \cdots + psk_n$. Any $t-1$ players possess, by default, $t-1$ pre-secret keys. However, they cannot compute the psk of any other player. Meanwhile, any t players can pool their shares to compute all the $psks$. These calculations are done in the same manner as Shamir's scheme [32].

We can also observe that, if F is the sum of all polynomials P_i, then $SK = F(0)$. Therefore, if t players are required to recover SK, they do not need to compute every polynomial $P_1, \ldots, P_n$ on route. Instead, players can first compute the sum of their shares—which is computationally inexpensive—then use these shares to compute a single polynomial, F, before recovering SK.

Generating the information required for frontloading in a decentralised manner

In the following, t players must collaborate to use the shared secret and t' players must collaborate to issue a new share. Each player has one share of the secret but this can be increased if desired. There are n initial players.

Initial setup

1. Each player generates a pre-secret key, denoted psk_i and a polynomial of degree $t-1$, denoted

$$P_i(x) = \sum_{j=0}^{t-1} \alpha_{i,j} x^j$$

   where $\alpha_{i,0} = psk_i$ and all other coefficients are random.

2. Each player then computes $n-1$ shares of their psk — i.e. $P_i(1), \ldots, P_i(n-1)$ — and distributes one share to each of the $n-1$ other players.

3. For $n \leq j \leq u$, each player also computes $P_i(j)$. These are the shares of the pre-secret keys which players can be given in the future.

4. For $n \leq j \leq u$, each player generates a polynomial of degree $t'-1$, denoted

$$Q_{i,j}(x) = \sum_{k=0}^{t'-1} \beta_{i,j,k} x^k$$

   where $\beta_{i,j,0} = P_i(j)$ and all other coefficients are random.

5. For $n \leq j \leq u$, each player computes $n-1$ subshares of $P_i(j)$ — i.e. $Q_{i,j}(1), \ldots, Q_{i,j}(n-1)$ — and distributes one share to each of the other initial players.

6. Each player computes their share of the secret where $F(j) = \sum_{i=1}^{n} P_i(j)$ is the share of player j.

Issuing a share

1. To issue a share, t' players select a previously unassigned j and, for $1 \leq i \leq n$, send their subshares of $P_i(j)$ to the recipient of the share. In other words, player i would send $\{Q_{1,j}(i), \ldots, Q_{n,j}(i)\}$ to the recipient of the share.

2. Once t' players have done this, the recipient has t' shares of each $P_i(j)$ for $1 \leq i \leq n$ and, since the polynomial used to distribute shares of each $P_i(j)$ is of degree $t'-1$, the recipient can calculate $P_i(j)$ for $1 \leq i \leq n$. This calculation is done in the same manner as Shamir's scheme [32].

3. Finally, the recipient can calculate the sum $F(j) = \sum_{i=1}^{n} P_i(j)$ which is a distinct share of SK.

More layers of secret sharing

Using a similar approach to above, we can define further layers of subshares. Thus, players can participate in issuing shares even when there are multiple degrees of separation between them and the players who generated the secret.

With three layers of secret sharing and n initial players, an initial player, player w, would receive the following:

1. the set $\{P_i(w) : 1 \leq i \leq n\}$ — i.e. one share of each psk_i — which can be combined to make a share of SK.

2. the set $\{Q_{i,j}(w) : 1 \leq i \leq n, n < j \leq u\}$ — i.e. one share of each $P_i(j)$ for $1 \leq i \leq n$ and $n < j \leq u$ — where the subset $\{Q_{i,j}(w) : 1 \leq i \leq n, j$ is previously unassigned$\}$ can be given to a new player, player j, when they join the game. With similar subsets from t' players, player j can recover $P_i(j)$ for $1 \leq i \leq n$ and, subsequently, recover a share of SK.

3. the set $\{R_{i,j,k}(w) : 1 \leq i \leq n, n < j \leq u, n < k \leq u\}$ where $R_{i,j,k}(x)$ is a polynomial of degree $t' - 1$ defined as

$$R_{i,j,k}(x) = \sum_{l=0}^{t'-1} \gamma_{i,j,k,l} x^l$$

   where $\gamma_{i,j,k,0} = Q_{i,j}(k)$ and all other coefficients are random. In other words, player w has one share of each $Q_{i,j}(k)$ for $1 \leq i \leq n$ and $n < j, k \leq u$.

The subset $\{R_{i,j,k}(w) : 1 \leq i \leq n, k < j \leq u, k$ is previously unassigned$\}$ can be given to a new player, player k, when they join the game. With similar subsets from t' players, player k can recover $Q_{i,j}(k)$ for $1 \leq i \leq n$ and $k < j \leq u$.

In turn, the subset $\{Q_{i,j}(k) : 1 \leq i \leq n, j$ is previously unassigned$\}$ can be given to a new player, player j, when they join the game. With similar subsets from t' players, player j can recover $P_i(j)$ for $1 \leq i \leq n$ and, subsequently, recover a share of SK.

B Modelling and Verification of Security Properties for Remote Registration Solutions

We have developed models for the three solutions: Duplicate, Proxy and Ring authenticators. The models have been developed in the symbolic model of cryptography using the automated protocol verifier ProVerif [14,15]. 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. 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, 26]. 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 [13–15, 19] for a further discussion on the topic of formal verification in the symbolic model and ProVerif.

We have considered three security properties. Namely, for each user, all of their authenticators registers the same list of public keys (Property 1); any of their authenticators can login after any other authenticator has registered to a RP (Property 2); and no illegitimate authentications can occur, that is, an adversary cannot reuse old messages to execute a successful authentication (Property 3). ProVerif is able to prove that all three of our solutions meet all three security properties. Properties 1 and 2 are related to design goals 1, 2, 7, 8 (Sec. 3.3). Property 3 is related to design goal 6 (also 7 and 8).

The repository with the models is available at [7], and we provide the most important details for them below.

Equational Theories

We require two equational theories for the three solutions, namely, signatures and ring signatures, as described in Sec. 3.2. To accomodate probabilistic signatures, it is customary to define an additional function symbol that handles the randomness. Therefore, for the case of signatures from Sec. 3.2, the equational theory consists of the function symbols $\mathsf{Pk}/1$, $\mathsf{Sign}/2$, $\mathsf{InternalSign}/3$, and $\mathsf{Verify}/3$, so that the ProVerif code for the signing function looks as follows:

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

Hence, the corresponding rewrite rule in the destructor is

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

Similarly, for the case of ring signatures from Sec. 3.2, we also define a probabilistic scheme with function symbols $\mathsf{Pk}/1$, $\mathsf{RingSign}/3$, $\mathsf{InternalRingSign}/4$, and $\mathsf{RingVerify}/3$. Moreover, we define a type for the list of public keys through the data constructor $\mathsf{PkList}(y_1, \ldots, y_n)$. Therefore, we have

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

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 $L = \mathsf{PkList}(\mathsf{Pk}(x_1), \ldots, \mathsf{Pk}(x_n))$, we have

$$\mathsf{RingVerify}(L, m, \mathsf{InternalRingSign}(x_1, L, m, r)) \;=\; \texttt{true}$$
$$\vdots$$
$$\mathsf{RingVerify}(L, m, \mathsf{InternalRingSign}(x_n, L, m, r)) \;=\; \texttt{true}.$$

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 conveniently this operation. See Sec. B below.

Finally, we also model the theories defined in Sec. 3.5. 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:

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

Note that due to technical constraints of the resolution algorithm, this equation has to be defined as presented above. That is, the algorithm will not work if the LHS and RHS of the equation are swapped.

Events and Security Properties

Recall that 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 allow to define security properties. Events can be thoughts as "local computations" or as mere check points on locations of interest that record object values at these locations. We prepend "A_" to all events executed at an authenticator process, and "R_" to all events executed at a RP process. We follow a naming convention similar to [24] 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 events are parametrized with 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 in the authentication. For example,

$$\mathsf{A_Running}(\; \underbrace{uid, \quad i, \quad x_i, \quad L_j,}_{\text{Authenticator identity terms}} \quad \overbrace{RP_j,}^{\text{RP identity term}} \quad \underbrace{p}_{\text{Agreement parameters}} \;).$$

Of course, not all events make use of all the terms. For example, the RP does not have access to the user identifier *uid*, authenticator index i or private key x_i, and therefore these parameters are omitted for its events.

Using these events, we are now in position to describe a number of high-level security properties. The main focus is to verify correct authentication between any registered authenticator in a given set belonging to a user with identifier *uid*. The notation $\mathsf{Event}(p_1, \ldots, p_n)@t$ signifies that event Event was executed with parameters $p_1, \ldots, p_n$ at time t. Although, in order to avoid ambiguity, we are expressing the security properties as fully expanded first-order logic formulas, ProVerif provides a simplified syntax to express them in a more concise way.

Sanity check

It is reachable that all the authenticators belonging to a user *uid* cast the same list of public keys to register against RP_j.

$$\begin{aligned}
&\exists uid, x_1, \ldots, x_n, k, RP_j, t_1, \ldots, t_n.\\
&\quad \mathsf{A_Register}(uid, 1, x_1, \mathsf{PkList}(\mathsf{Pk}(x_1), \ldots, \mathsf{Pk}(x_n)), RP_j)@t_1 \wedge\\
&\quad\quad \vdots\\
&\quad \mathsf{A_Register}(uid, n, x_n, \mathsf{PkList}(\mathsf{Pk}(x_1), \ldots, \mathsf{Pk}(x_n)), RP_j)@t_n.
\end{aligned}$$

Property 1

For a given authenticator set belonging to user *uid* and RP_j, all authenticators of *uid* will register the same list of public keys.

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

Property 2

Is it reachable that an authenticator belonging to user *uid* registers all the user's authenticators, and any other authenticator of *uid* can then log in. We present an instantiation of this property for Authenticator 1, showing that Authenticator 1 can register with RP_j and all the other authenticators can authenticate. ProVerif finds a trace in which only Authenticator 1 registers, which is the trace we want. (We could have added add restrictions preventing the other authenticators from registering, but this was unnecessary since the trace that ProVerif finds already has that property).

$$\begin{aligned}
&\exists uid, x_1, \ldots, x_n, k, RP_j, p_1, \ldots, p_n, t, t_1, \ldots, t_n, t'_1, \ldots, t'_n.\\
&\quad \mathsf{A_Register}(uid, 1, x_1, L_j, RP_j)@t \wedge\\
&\quad \mathsf{A_Running}(uid, 1, x_1, L_j, RP_j, p_1)@t_1 \wedge\\
&\quad\quad \mathsf{R_Commit}(RP_j, L_j, p_1)@t'_1 \wedge\\
&\quad\quad\quad \vdots\\
&\quad \mathsf{A_Running}(uid, n, x_n, L_j, RP_j, p_n)@t_n \wedge\\
&\quad\quad \mathsf{R_Commit}(RP_j, L_j, p_n)@t'_n.
\end{aligned}$$

Similar formulas showing a trace in which only Authenticator i registers and all the others authenticate can also be established ($i > 1$).

Finally, the last property corresponds to a variation of injective agreement [24] for registered authenticators. This is the main property we want to test.

Property 3

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.

$$
\begin{aligned}
&\forall uid, i, L_j, RP_j, p, t_1, t_2, t_4.\\
&\quad \mathsf{A_Register}(uid, i, x_i, L_j, RP_j)@t_1 \wedge\\
&\quad \mathsf{R_Register}(RP_j, L_j)@t_2 \wedge\\
&\quad \mathsf{R_Commit}(RP_j, L_j, p)@t_4 \Rightarrow\\
&\qquad \big(\exists i', x_{i'}, t_3.\ \mathsf{A_Running}(uid, i', x_{i'}, L_j, RP_j, p)@t_3 \wedge (t_3 < t_4)\big) \wedge\\
&\qquad \neg\big(\exists t_4'.\ \mathsf{R_Commit}(RP_j, L_j, p)@t_4' \wedge \neg(t_4 = t_4')\big).
\end{aligned}
$$

Indeed, the events A_Register and R_Register indicate that there has been a successful registration of the authenticators belonging to user uid. The event R_Commit indicates a successful authentication at the RP side. Therefore, if all these three events are encountered in a trace, it must be the case that a legitimate authenticator executed the event A_Running before the RP executes 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 o a distinct run of the protocol by the authenticator.

Macro Language

In order 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 pre-specified number of repetitions, in our case the number of authenticators per user n. The construct is as follows:

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

where ⟨`prefix`⟩ and ⟨`suffix`⟩ are constant, arbitrary strings, ⟨`var_name`⟩ is the name of the variable to be substituted, and ⟨`separator`⟩ is the separator string to be interleaved between substitutions (a default value of "`, `" is taken if not specified). This construct is required, to define queries and prototypes of equational theories, where the user can later specify the number of authenticators to include in the verification process.

As an example of its syntax, consider the following construct that defines a list L of an unknown number of public keys $y_1, \ldots, y_n$:

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

which for three authenticators ($1 \leq i \leq 3$) will expand to

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

by calling the macro preprocessor with the corresponding input arguments. The syntax also allows to define 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.

Although arguably the cases of interest will be typically $n = 2, 3$, we believe that it is important to have evidence that the proposed solutions are secure, independently of the number of authenticators chosen. For that purpose, the macro language has allowed us to use ProVerif to test that the security properties specified in Sec. B above hold for the three Solutions presented, for several scenarios with different number of authenticators.

References

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

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

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

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

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

[6] FIDO Alliance. 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`, 2021.

[7] Anonymous submission. Google Drive repository for ProVerif models, October 2022. `https://tinyurl.com/2p8ehbf2`.

[8] Myrto Arapini. *Security of Cryptographic Protocols: Decidability and Reduction Results*. PhD thesis, 2008.

[9] Columbia University Asia for Educators. The song economic revolution. `http://afe.easia.columbia.edu/songdynasty-module/econ-rev-money.html`, 2022.

[10] Boris Balacheff, Gaetan Wattiau, Joshua Schiffman, and Thalia M Laing. Multiple device-based authentication. HP Labs Disclosure ID 85725461, 2019.

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

[12] Leo Benedictus. A brief history of the passport. `https://www.theguardian.com/travel/2006/nov/17/travelnews`, 2006.

[13] B. Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 1(1-2):1–135, 2016.

[14] B. Blanchet, V. Cheval, and M. Sylvestre. ProVerif. `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`.

[15] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre. *ProVerif 2.04: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, November 2021.

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

[17] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Provably authenticated group diffie-hellman key exchange—the dynamic case. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 290–309. Springer, 2001.

[18] World Wide Web Consortium. Web authentication: An API for accessing public key credentials level 2 W3C recommendation. `https://www.w3.org/TR/webauthn/`, 2021.

[19] V. Cortier and S. Kremer. Formal models and techniques for analyzing security protocols: A tutorial. *Foundations and Trends in Programming Languages*, 1(3):151–267, 2014.

[20] Thalia M Laing and Joshua Schiffman. Securely introducing new devices into a threshold scheme. HP Labs Disclosure ID 700236841, 2019.

[21] Thalia M Laing and Douglas R Stinson. A survey and refinement of repairable threshold schemes. *Journal of Mathematical Cryptology*, 12(1):57–81, 2018.

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

[23] LastPass. About password iterations. `https://support.lastpass.com/help/about-password-iterations-lp030027`, 2022.

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

[25] 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.

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

[27] Mehrdad Nojoumian, Douglas R Stinson, and Morgan Grainger. Unconditionally secure social secret sharing scheme. *IET information security*, 4(4):202–211, 2010.

[28] The Editors of Encyclopaedia Britannica. Lock. `https://www.britannica.com/technology/lock-security`, 2021.

[29] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 522–526. Springer, 1991.

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

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

[32] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[33] Wikipedia. World wide web consortium. `https://en.wikipedia.org/wiki/World_Wide_Web_Consortium`, 2022.