Symbolon: Enabling Flexible Multi-device-based User Authentication

Thalia Laing
HP Labs
Bristol, UK
thalia@hp.com

Eduard Marin*
Telefonica Research
Barcelona, Spain
eduard.marinfabregas@telefonica.com

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

Joshua Schiffman
HP Labs
Bristol, UK
joshua.ser.schiffman@hp.com

Gaëtan Wattiau‡
Decentriq
Zürich, Switzerland
gaetan.wattiau@decentriq.ch

***Abstract*—Hardware tokens are increasingly used to support second-factor and passwordless authentication schemes. While these devices improve security over weaker factors like passwords, they suffer from a number of security and practical issues. We present the design and implementation of Symbolon, a system that allows users to authenticate to an online service in a secure and flexible manner by using multiple personal devices (e.g., their smartphone and smart watch) together, in place of a password. The core idea behind Symbolon is to let users authenticate only if they carry a sufficient number of their personal devices and give explicit consent. We use threshold cryptography at the client side to protect against strong adversaries while overcoming the limitations of multi-factor authentication in terms of flexibility. Symbolon is compatible with FIDO servers, but improves the client-side experience compared to FIDO in terms of security, privacy, and user control. We design Symbolon such that the user can (i) authenticate using a flexible selection of devices, which we call "authenticators"; (ii) define fine-grained threshold policies that enforce user consent without involving or modifying online services; and (iii) add or revoke authenticators without needing to generate new cryptographic keys or manually (un)register them with online services. Finally, we present a detailed design and analyse the security, privacy and practical properties of Symbolon; this includes a formal proof using ProVerif to show the required security properties are satisfied.**

***Index Terms*—authentication, FIDO, threshold cryptography, signatures, proverif**

I. Introduction

The security and usability issues plaguing passwords as a form of user authentication on the web are well known [2], [23]. To compensate for these weaknesses, security professionals [18] advocate complementing or even replacing passwords with strong cryptographic factors rooted in hardware-based security mechanisms. Solutions like portable hardware tokens or integrated security micro-controllers offer many advantages as a form of authentication over passwords, one notable advantage being that hardware tokens, unlike users, are able to memorise and use cryptographically strong keys. The Fast IDentity Online (FIDO) Alliance [14] has recently produced a standard framework that aims to create a common and easy-to-use ecosystem for users to authenticate online through hardware-based security keys. FIDO has been adopted by a number of companies, such as Google, Dropbox and Facebook, and is now supported by Microsoft and OpenSSH.

Despite FIDO-like protocols providing many benefits over memorised secrets, they still suffer from limitations [22]. From a security standpoint, hardware tokens are vulnerable to theft. On one hand, if a token is used as the sole authentication factor, a thief can steal the token and use it to impersonate them. While asking users to locally authenticate to the token first (e.g., using a PIN or biometrics) can mitigate these threats to some extent, these devices tend to offer limited (or no) protection against adversaries who have physical access to them (see Section IX for more details). On the other hand, if the hardware token used as a second factor is stolen, the user is left with the remaining first factor to protect their account, which is normally much weaker than the token. In short, although hardware tokens enable users to protect their accounts using cryptographic keys, the risk of loss of the hardware token (and hence the whole key) must be considered.

Besides these security concerns, FIDO has several practical limitations. For example, the user must have a hardware token available whenever they wish to authenticate. Forgetting, losing, or damaging the token will render the user unable to access their account, as their token is a single point of failure in terms of availability[1]. To reduce this frustration, FIDO (and other similar protocols) allows users to register multiple tokens for a single user account, and authenticate with any of them. This solution increases the availability of authentication for the user by having more tokens, but in doing so it (i) gives a physical attacker multiple devices to target for theft and (ii) allows the online service to learn potentially sensitive information about which of their devices a user authenticates

*Work done while at University of Birmingham
†Work done while appointed as HP Research Chair at HP Labs
‡Work done while at HP Labs

[1]Note that this issue is aggravated by the fact that users typically reuse their authenticators across websites.

with[2]. Besides, adding alternative tokens in this way also places a burdensome management process on the user. If one of their tokens is lost, broken or replaced, the user will need to unregister and register new tokens with each online service they have accounts with. This is a manual and laborious task that should be avoided.

We present Symbolon, a multi-device based user authentication solution that increases the resiliency of FIDO-like systems against physical attackers whilst simultaneously increasing the availability of the system and allowing easy management for users. At a high level, Symbolon uses a threshold signature scheme to register a *single* public key to an online service, and secret-shares the private key among a set of user's personal devices. The user is then able to authenticate only if a threshold number of their personal devices are present, and after giving explicit consent. Symbolon extends solutions such as FIDO and requires changes on the client side only, meaning that FIDO compatible servers are not required to be updated.

In summary, Symbolon is compatible with FIDO servers, but improves the client-side experience compared to FIDO in these ways:

- **Security.** Compromise of a single hardware token authenticator does not allow the adversary to masquerade as the user (as it can in FIDO). Thus, adding new authenticators does not introduce single points of security failure.
- **Flexibility, privacy, and user control.** Adding/removing authenticators is done independent from the online services. The user can authenticate with flexible subsets of their devices, the online services do not know which or how many authenticators the user has, nor which ones they use in any particular authentication session.

Our contributions. In this paper, we propose Symbolon, a framework allowing multiple personal devices chosen by a user to contribute to user-authentication sessions, in order to make user authentication more secure while maintaining flexibility for the user. Concretely, the contributions of this work are the following:

- We detail Symbolon; conceptually quite simple, there are tricky issues that arise when figuring out the details. Resolving these is our principal contribution.
- We present Symbolon iteratively, first by means of an example (Section III), and then informally focusing on the requirements (Section IV). Next, we give the formal details (Section V), including how the user selects devices, uses them to authenticate, and updates their selection with new or lost devices.
- We extend Symbolon to minimise the need for explicit user consent on each device while cryptographically enforcing consent on at least one user-specified authenticator device (Section VI).
- We provide measurements of the feasibility of our solution in terms of computational load on devices and speed of authentication, derived from our implementation (Section VII).
- We analyse the security, privacy and practicality of our solution (Section VIII). This includes a formal evaluation of security properties using ProVerif (see [4]).

A longer version of the paper containing some extra material is available at https://www.dropbox.com/sh/rh0n9qm1gnf7k0y/AAAAcx8_0Ayw9YGEdYnQhEmda?dl=0 [4].

[2]Online services may accumulate information about which device is used for each authentication session to infer information about the user, such as where they are, whether their routine is changed, or whether a device is lost.

II. Background

FIDO authentication. Figure 1 illustrates a generalised architecture of FIDO-based authentication schemes. Users interact with online services, called *relying parties (RP)*, through a *client device* like a laptop or mobile phone. The RP authenticates the user using an (internal or external) authentication service. The user possesses a FIDO *authenticator* that protects the user's identity keys and performs the cryptographic operations needed to authenticate the user. An authenticator may be internal (e.g., secure element or Trusted Platform Module) or external (e.g., a USB token) to the client device.

FIDO-based authentication over the web is defined by two specifications: Web Authentication (WebAuthn) [35] and the Client to Authenticator Protocols (CTAP) [3]. The former is a standard web API that can be built into existing browsers, while the latter enables authenticators to interact with browsers supporting WebAuthn. FIDO authentication is realised by means of a challenge-response protocol that comprises two phases: *registration* and *authentication*.

Registration phase. Initially, the user plugs their authenticator into a client device and sends a request to an RP through a browser or app in the client device. The RP generates a random $challenge$ and returns it to the user's browser over a secure channel (such as TLS). The browser sends the user's authenticator a hash of the client data (denoted c), which includes: (i) $origin$ (RP's URL), (ii) the $challenge$, and (iii) an optional $binding$ (to preclude MiTM attacks). The user's authenticator then generates a public/private key pair, k_{pub} and k_{priv}, and a key handle H_k to retrieve k_{priv} during the authentication phase. Next, the authenticator returns k_{pub} and H_k along with a signature over c, k_{pub} and H_k to the browser. This information is forwarded along with the $challenge$ and $binding$ to the RP. Finally, the RP verifies the signature using k_{pub} and stores k_{pub} and H_k.

Authentication phase. When the user wants to authenticate to an RP, they send a request containing the account they wish to authenticate to. The RP then checks whether there exists an H_k associated with the user's account. If H_k is found for this user's account, the RP generates a random challenge and sends it along with H_k to the user's browser. The browser then sends the authenticator: (i) the $origin$, (ii) H_k and (iii) the hash of the client data c. Upon receiving this information, the user's authenticator verifies whether (i) it has a k_{priv} associated with H_k and (ii) it received an origin that matches the one stored alongside H_k. If both conditions are satisfied, the authenticator waits for the user to give explicit consent (e.g., by pressing a button on it) to authenticate to the RP. Only if consent is

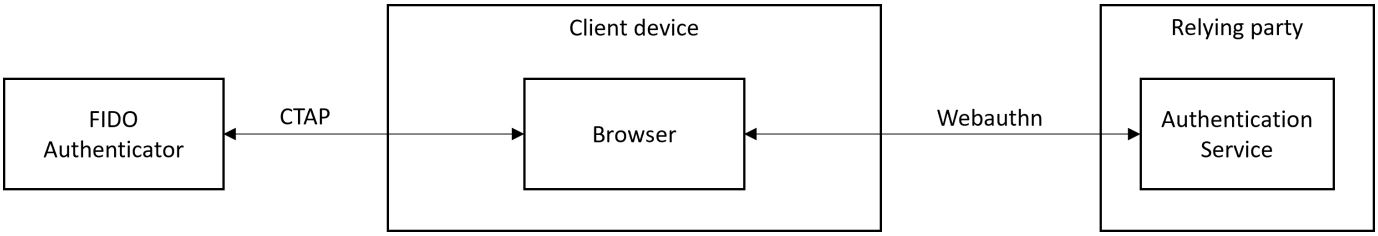

Fig. 1: Simplified FIDO architecture.

given will the user's authenticator unlock k_{priv} in order to produce a signature on c and send this information to the browser. The browser forwards the $signature$, the $challenge$ and the $binding$ to the RP. Finally, the RP verifies the received signature using k_{pub} stored in relation to the user's account.

Threshold secret sharing schemes distribute a secret s by a *dealer* amongst a set of entities, by giving each entity a *share* of the secret [6], [28]. In a (t, n)-threshold scheme, where $t, n \in \mathbb{Z}, 2 \leq t \leq n$, s is distributed between n entities such that any subset of t can jointly recover s. Shamir's scheme [28] distributes s by defining a polynomial $f(x)$ of degree $t-1$ with random coefficients such that $f(0) = s$. Entity i for $1 \leq i \leq n$ is allocated the share $f(i)$. The secret can be reconstructed by t entities performing polynomial interpolation, whilst fewer than t entities learn no information about s.

Threshold signatures use secret sharing to construct a signature scheme with n signers, where a valid signature can only be produced if at least t signers participate in the signing. There exist threshold signatures compatible with existing verification algorithms, including RSA [11], [29], DSA and ECDSA [16], [17] and Schnorr [33]. Some threshold signatures, such as Shoup's RSA scheme [29], are non-interactive and each signer outputs their contribution to the signature, which we call a *partial signature*, which are combined to produce the complete signature. Other schemes, such as [16], [17], are interactive and require rounds of communication between signers.

III. Motivating example

FIDO-based authentication offers users the ability to authenticate with a simple physical token, greatly improving security over weaker, memory-based passwords. However, the reality of authenticating with a physical token is that a user may lose it (leading to loss of availability to the account), or have it stolen or compromised by an attacker. Even if the token requires users to locally authenticate, attackers with physical access can launch practical and effective side-channel attacks to bypass the local authentication step and gain access to the user's account (see Section IX for more details).

Let us now consider a more convenient and secure system for a user, Alice, who wants to authenticate to her bank using a FIDO-based protocol. Without loss of generality, we assume Alice starts by enrolling four devices (her laptop, phone, smartwatch and Bluetooth headphones) as authenticators. She then specifies that at least three of these devices must successfully authenticate before the bank is convinced it is Alice. In many online services today, registering multiple devices is possible, as is requiring multiple factors for successful authentication. However, requiring a threshold policy (such as any three of the four authenticators) is not generally possible. Even if it were possible, Alice would need to rely on the RP to both implement and enforce such a policy. Due to this, the RP might also begin to infer access patterns based on which devices Alice chooses to use at each authentication session.

To address these issues, Symbolon employs threshold signatures and shares the signing capability of a FIDO authenticator among Alice's devices. This offers more user-centric control over the use and management of a multi-device-based authentication experience. With Symbolon, Alice uses a *dealer* to generate and enrol a public key to the RP, and to share the corresponding private key amongst her four devices. The dealer is trusted to protect the private key and is only used to enrol to new RPs and update the set of participating devices. In Section VII we detail possible ways to realise the dealer in practice. In brief, the dealer can be either a physical device (distinct from the authenticators), or an online cloud service. Say Alice uses a USB token as her dealer. She plugs the dealer into her laptop and interacts it with via the dealer's GUI.

Now that Alice has completed the set-up, she can store her dealer token in a secure location and use any three of her authenticators to authenticate to the bank. When she tries to log in to her bank account using her laptop, the browser receives the challenge from the bank and distributes it to all available authenticators. The authenticators partially sign the challenge using their key share; some will also capture Alice consenting to authenticate, e.g., by pressing a button on a device (see Section VI). Subsequently, the browser combines these partial signatures and forwards the result to the bank. The signature will be valid and therefore the authentication to the bank successful provided a threshold number of partial signatures are combined. The result is an authentication experience that enables flexible choice of authenticators and threshold policies without the need for Alice to depend on the bank to implement such a mechanism using more traditional multi-factor authentication flows that also discloses her device usage patterns.

At any future time, Alice can edit the set of registered authenticators. She might do so if she loses her phone, for example. To do this, she retrieves her dealer and interacts with the dealer's GUI to add or remove authenticators.

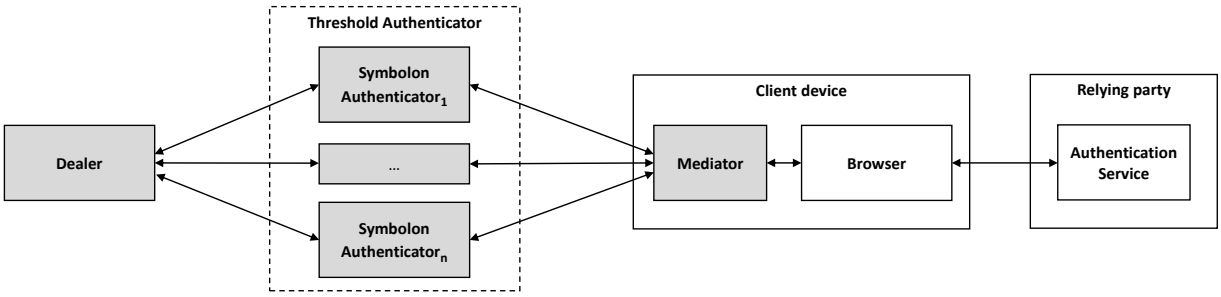

Fig. 2: Symbolon extends the FIDO architecture to support threshold-based authenticators. Shading indicate new components.

IV. System design

We now describe the design of Symbolon including our design requirements, assumptions, and threat model.

A. Architecture

We designed Symbolon to enable threshold secret sharing of a single authenticator's signing key. Figure 2 illustrates how our architecture extends the FIDO2 model to achieve this. This includes the following changes:

Symbolon authenticators. Instead of a single authenticator, a collection of the user's personal devices – such as a smartphone, laptop or smart watch – work together as a collective "threshold" authenticator. A dealer provisions individual key shares to each authenticator per account. The authenticators coordinate with a mediator to perform partial signing operations that contribute to a full signature during authentication. Each device is capable of storing data, performing cryptographic operations and communicating with the dealer and mediator over a secure communication channel. Some authenticators (e.g., smartphones) are capable of secure user I/O, while others (e.g., spectacles) may have limited or no user interface.

Dealer. The dealer is responsible for generating user authentication credentials (signing key pair) and distributing shares of the private key to each of the authenticators. We envision the dealer as a secure, dedicated service or a device specific to and controlled by the user. The dealer is only needed during registration with an RP and when modifying the set of authenticators. It is not involved in authentication phases and thus it is not an authenticator the user employs to authenticate.

Mediator. The mediator acts as a bridge between the Symbolon authenticators and the browser or app in the client device. It forwards challenges from the RP to the authenticators, collects partial signatures, and combines them to form a response. The mediator is stateless and can be on multiple client devices.

Our design effectively decouples the registration and authentication capabilities of a single authenticator, giving the user the flexibility to manage their set of authenticators independent from the RP. The user also benefits from the resiliency of the threshold scheme as adversaries must compromise multiple devices to authenticate as the user. Finally, the user can leverage more hardened but less convenient key protection mechanisms for the dealer as it is not required for authentication. There are several ways the dealer can be realised to maximise security and availability, discussed further in Section VII.

Authenticators must be securely paired with the dealer to protect the distribution of key shares. Different options may be supported according to the capabilities of those devices. For example, pairing protocols like Bluetooth's LE Secure Channels mode with Numeric Comparison gives strong MITM protection with user verification [7]. In other scenarios, pre-installed identity certificates or out-of-band channels may suffice depending on the user's concerns. The latter is an attractive solution that allows pairing of devices with limited interfaces. For example, if the devices have (at least) a standard input and/or output interface, it is possible for the user to pair the devices by manually copying the data output from one device into the other, comparing the output of both devices, or entering the same data into both devices [15]. Likewise, the user can pair the devices by shaking them simultaneously in case they have no I/O interfaces but are equipped with an accelerometer [10].

B. Design requirements

Symbolon aims to satisfy the following security, privacy and practical requirements.

(Sec1) Threshold-based security. Anyone with fewer than the predefined threshold number of authenticators should not be able to authenticate as the user to an RP.

(Pri1) Authenticator privacy. An RP should not be able to learn the number of authenticators a user has, the threshold number of authenticators required, whether authenticators are added or revoked, or distinguish between different sets of authenticators used to authenticate at different times.

(Pra1) Flexibility. Users should be able to authenticate successfully to an RP on any client device using any t of their authenticators.

(Pra2) User control. Authenticated users should be able to set, modify, implement and enforce their own threshold policies without involvement of the RP. Furthermore, Symbolon should guarantee that successful authentication to an RP occurs only if users explicitly provide consent.

(Pra3) Authenticator set flexibility. Users should be able to revoke/add authenticators easily without needing to generate

new cryptographic keys or manually (un)register them with RPs. Furthermore, the computation and communication cost involved from the remaining authenticators due to these procedures should be minimal.

(Pra4) FIDO2/WebAuthn compatible. Symbolon should be compatible with existing browsers and RPs that authenticate users using asymmetric signing keys (e.g., FIDO-compliant services).

C. Threat model and assumptions

Threat model. Symbolon defends against adversaries whose goal is to masquerade as the user to an RP by stealing, compromising or leveraging genuine contributions. In particular, we adopt the Dolev-Yao adversary model [12] where adversaries can eavesdrop, intercept and modify traffic as well as relay signals transmitted over the air. Adversaries can also compromise both up to $t-1$ user's authenticators (by, for example, physical theft, malware running on the device or remote code execution) and servers that do not implement strong security practices. We model RPs as *honest-but-curious*, meaning they follow the protocol as intended but try to learn information about how users authenticate, e.g., which set of devices users employ to authenticate. Denial-of-Service attacks (e.g., blocking communication to an RP) are out of scope of our work.

Assumptions. We employ a centralised dealer that is assumed to be trusted. Such a dealer is analogous to a recovery key. Our choice is motivated by the need to give users the ability to easily (and securely) modify the set of authenticators or reduce the threshold. As demonstrated by Ghorban et al. [22], this is a fundamental requirement for user acceptance. Technically, this can also be achieved through a decentralised dealer [19], [25]. Yet, these solutions are less convenient for users and require all devices to collaboratively store, manage and display information, which introduces a significant communication overhead. The client device acts as a bridge between the user's authenticators and the RP. While the client device does not store any secrets and hence need not be trusted, it runs the mediator correctly, forwards the information exchanged between the aforementioned parties as intended and communicates with the RP using secure channels (e.g., TLS). For now, we assume a secure non-interactive threshold signature scheme in order to reduce the rounds of communication occurring over possibly slow channels. Our system could be adapted to use an interactive threshold scheme, but the mediator may need to help the authenticators interact during each authentication session. We assume the authenticators communicate with the dealer over a confidential, authenticated end-to-end channel and that user interfaces are secure.

D. Threshold parameter choices

The user must choose the initial parameters n and t to register with. The user may choose n to be the number of personal devices they have available to them for authentication and then, once n is chosen, consider t. When choosing t, there is a trade-off to be considered: the closer t is to n, the more the system is resilient to theft, but the less flexible the system is to the user losing or forgetting devices. As t decreases, flexibility supersedes resilience. The choice of t will also depend on what the devices are and how they are stored and used. Notably, if $t \leq n/2$, an adversary may be able to compromise t devices while the user is still in possession of a distinct set of t honest devices. The user may choose $t > n/2$ to avoid this happening. To help the user, the system may suggest some sensible default parameters, such as $n = 3$ and $t = 2$.

V. Protocol

Symbolon uses a non-interactive threshold signature scheme to let users authenticate to RPs. Partial signatures are combined on the client side, producing a complete signature that is sent to the RP. The signature is then verified using a standard verification algorithm, thus our system works with any FIDO2/WebAuthn-enabled server. Below, we describe each stage of the system.

Registration phase. The goal of the registration phase is for the user to select a set of their personal devices to act as their logical authenticator to the RP. The user must choose which of their personal devices will act as authenticators, and pair each device with the dealer. The procedure through which the dealer establishes a secure communication channel with the user's personal devices depends on the interfaces and capabilities they have (some options were discussed in Section IV-A).

Once the user's personal devices have enrolled with the dealer as authenticators, the user can initiate the registration procedure with an RP. Figure 3 shows the Symbolon registration phase. The dealer receives the *origin* value and a hash from the mediator; this message is the standard FIDO message from the browser to the authenticator, since we want to remain compatible with FIDO. Next, the dealer generates a key pair, k_{pub} and k_{priv}, and registers k_{pub} with the RP. According to parameters chosen by the user via a secure interaction (as discussed in Section IV-A), the dealer shares k_{priv} into n shares k_i, for $i \in (1, ..., n)$, and sends each authenticator their share k_i along with the origin (the DNS name of the RP), key handle H_k and (if necessary for combining partial signatures) n and t. Each authenticator stores this data, while the dealer stores H_k, $origin$, k_{priv} and the randomness generated to share k_{priv} (i.e., the $t-1$ coefficients of the polynomial in Shamir's scheme). We emphasise that the dealer keeps k_{priv} to offer a mechanism for users to revoke and add authenticators after the initial registration (discussed later).

The registration phase is executed once per user account. The user can follow this procedure to register more accounts with the same or other RPs. As each account uses a different public/private key, users can reuse their set of authenticators across RPs without parties being able to track their accounts.

Authentication phase. The purpose of this phase is to authenticate the user to an RP through a set of (at least) t of their authenticators. Our authentication protocol, shown in Figure 4, starts by the user connecting to the RP through their client device. Following similar steps to those employed by FIDO, the RP checks whether there exists an H_k associated

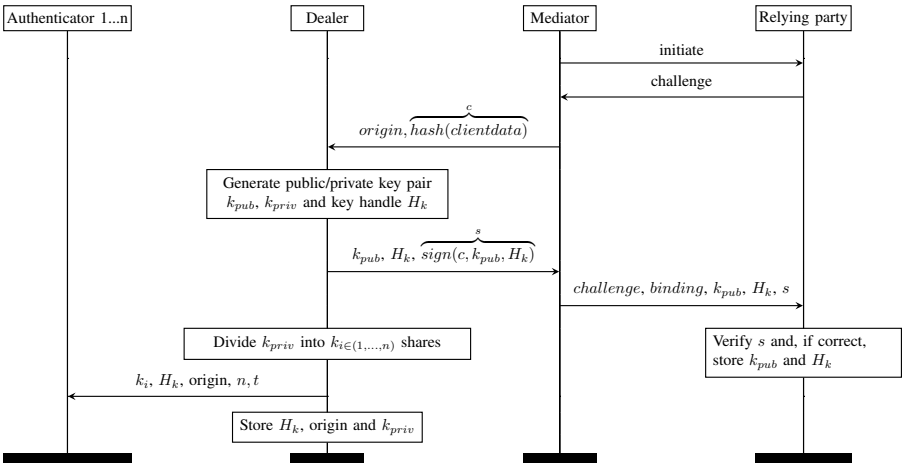

Fig. 3: Symbolon registration. Client data includes *origin*, *challenge*, and *binding*.

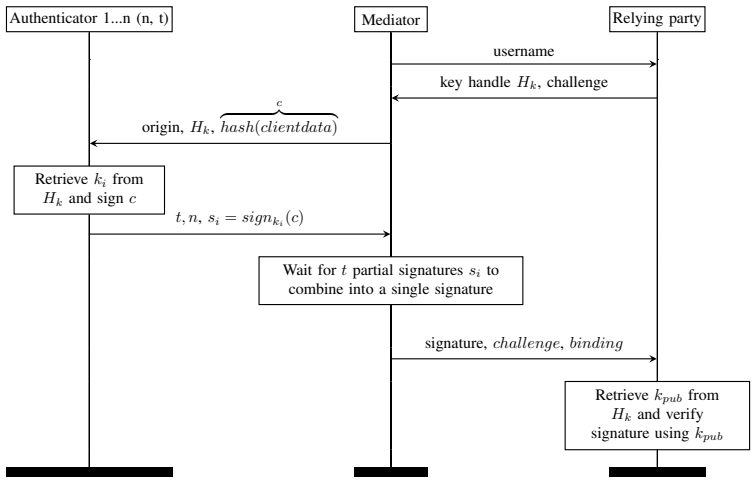

Fig. 4: Symbolon authentication. Client data includes *origin*, *challenge*, and *binding*.

with the user's account. Only if this condition is satisfied, the RP generates a random challenge and sends it along with H_k to the client device's browser (or App). This information is forwarded to the mediator, which broadcasts the hash of the *challenge* and $ChannelId$ along with H_k and *origin* over all its communication channels (e.g., USB or Bluetooth). Upon receiving the broadcasted message, each authenticator individually verifies whether (i) it has a share associated with H_k and (ii) it received an origin that matches the one stored alongside H_k. If both conditions are fulfilled, the authenticator computes a partial signature on the hash of the *challenge* and $ChannelId$ and returns their partial signature, along with t and n (if needed to compute the combination function) to the mediator. Beyond the initial consent expressed on the browser, our system requires the user to give consent on one of their authenticators in order to authenticate to the RP. (We refer the reader to Section VI for the details on how user consent is achieved and enforced.) Subsequently, the mediator waits to receive partial signatures from user's authenticators during a specified time window. Once this period expires, the mediator compares the values of t from each of the user's authenticators. Provided all authenticators send the same t value and at least t of them contribute, the mediator combines t partial signatures into a single signature. (If inconsistencies are detected in the information reported by the authenticators, the protocol is aborted.) The signature is then sent along with the *challenge* and $ChannelId$ via the browser to the RP. Finally, the RP retrieves k_{pub} from H_k and verifies the received signature.

Maintenance phase. At any point, the user can modify their set of authenticators by using the dealer. Symbolon allows the user to execute some of these operations (all, if forgetfulness on the authenticators can be enforced), without the user needing to authenticate to the RP and update k_{pub}.

Adding a new authenticator. Suppose the user buys a new device and wants to add it as an authenticator for one of their accounts. Initially, the user must enrol their device with

the dealer. Through the dealer's GUI, the user selects the account to which they want to add the device. At this point, the dealer computes a new share for the device by using k_{priv} and the stored randomness (by, for example, evaluating the stored polynomial in a Shamir threshold scheme at the point $n+1$). Then it sends the share along with H_k, $origin$, t (if necessary) and the newly updated $n+1$ to the new authenticator. Subsequently, when the authenticators send the parameters to the mediator to combine the partial signatures, the mediator will use the highest value of n received (the newly added authenticator will have the new parameter $n+1$, whilst the others will have n). If the user plans to use this new authenticator for multiple accounts, the dealer can provide a new share for each of the accounts the user selects simultaneously.

Revoking an authenticator. Users must also be able to revoke authenticators if they get lost or stolen. Our revocation mechanism leverages techniques from proactive secret sharing [19] to allow the revocation of up to $n-t$ devices without updating k_{pub}. The user first indicates on their dealer which authenticator they wish to revoke from which account. Next, the dealer computes and sends new shares to each of the remaining $n-1$ authenticators; these new shares are a new sharing of k_{priv} with fresh randomness. Upon receiving the new share, the authenticators *must* erase the old share and start to use the new share. As the revoked authenticator does not receive the update, the share it stores is incompatible with the new shares and can no longer be used to produce a valid partial signature. To avoid revoked authenticators collaborating and authenticating on behalf of the user, at most $t-1$ authenticators can be revoked simultaneously. This ensures t revoked authenticators will not have compatible shares and will not be able to maliciously authenticate.

If we cannot enforce the remaining authenticators to erase their old shares, we can remove at most $t-1$ authenticators before having to refresh the system and execute the registration phase again, otherwise the t revoked authenticators could collaborate and authenticate with their old shares.

Decreasing t. A user may want to decrease t if, for example, they initially set t to be too high for their convenience. As with adding and revoking an authenticator, this change can be made without updating the k_{pub} on the RP. To do this, the user indicates on the dealer that they wish to reduce t to, say, $t-1$. The dealer computes a new share using k_{priv} and the stored randomness and broadcasts this share along with H_k to all authenticators in the system. Every authenticator stores this share alongside its other share, and reduces its stored value of t. When the user authenticates, at least one authenticator must submit a partial signature using the additional share as well as a partial signature using their original share. Thus, $t-1$ devices know t shares between them (their original $t-1$ shares plus the additional share) and so can authenticate.

Increasing t. If we can enforce the authenticators forgetting old shares, increasing t can be achieved by the dealer simply re-sharing k_{priv} with new randomness, sending new shares to all authenticators, and all authenticators deleting past shares. If we cannot enforce authenticators erasing old shares, the user must execute the registration phase again and generate a new key pair with the new t. This is because, if we cannot enforce deletion of past shares, there is nothing to stop the authenticators using past shares, related to the lower threshold, to authenticate, thus bypassing the increased threshold.

VI. Enforcing user consent

So far Symbolon relies on a straightforward threshold signature scheme where all authenticators contribute equally to each user authentication instance. Now, we adapt it to ensure the user has actively granted consent to the authorisation on some set of authenticators, e.g., by pressing a button. We define a subset of authenticators that are able to register user consent and adapt our use of threshold signatures to ensure that some number of those devices (a 'consent threshold' t_c) participate in the protocol. Rather than simply distributing a signing key amongst t authenticators as we did in Section V, we share the signing key using iterative threshold schemes (first done by Ito et al. [20]). In order to use such constructions to achieve our desired access structure, we first split the signing key using a $(2,2)$-threshold scheme, then distribute the first share amongst consentful authenticators, and use a (t,n)-threshold scheme to distribute the second share amongst all authenticators. In practice, this works as follows.

Setup phase. The user enrols their devices with the dealer as in Section V, but the dealer additionally learns whether each device is consentful (i.e., can register user consent). This information is stored. The dealer then initiates registration with an RP and proceeds as before until k_{priv} is distributed. Rather than sharing k_{priv} into n shares, the dealer computes the following steps:

- Distribute k_{priv} using a $(2,2)$-threshold scheme. Denote the resulting two shares of the key output as k_c and k_a. (For example, if an RSA signature is used, a $(2,2)$-threshold scheme can be executed by generating k_c and k_a such that $k_c + k_a = k_{priv} \mod \phi(N)$, where N is the RSA modulus, as in Section 3 of [9].)
- Distribute k_c amongst the consentful authenticators according to the parameters (t_c, n_c), where n_c is the total number of consentful authenticators and t_c is the 'consent threshold' (i.e. the number of authenticators the user must register consent on, with t_c chosen by the user such that $1 < t_c \leq t$, we anticipate $t_c = 1$ in most cases). Both k_c and the randomness generated to distribute k_c are stored by the dealer.
- Distribute k_a amongst all n authenticators according to the parameters (t,n). Both k_a and the randomness generated to distribute k_a are stored by the dealer.
- The dealer sends each authenticator its share of k_a, H_k, $origin$, t and n. If the authenticator is consentful, it is also given a share of k_c and parameters t_c, n_c.
- As before, the dealer stores k_{priv}, and now also k_a and k_c (along with the randomness used to distribute each value).

Authentication phase. Following the registration procedure, the user's authenticators first receive a broadcasted message

corresponding to a stored $origin$ and H_k, then sign the message with their share of k_a and return their partial signature along with the necessary parameters. In addition, any consentful authenticators will (if possible) display a message to the user requesting consent. If consent is registered, the consentful authenticator will send their partial signature using their share of k_c along with any necessary parameters (t_c, n_c) to the mediator, and a flag marking this as a consentful partial signature. Once the mediator receives sufficiently many partial signatures (t partial signatures and t_c partial signatures flagged as consentful), the mediator will:

- Combine all consentful partial signatures to create a signature on the message using k_c.
- Combine all the remaining (non-consentful) partial signatures to create a signature on the message using k_a.
- Combine the two signatures to create a complete signature on the message.
- Pass the signature to the browser, who passes it to the RP.

Note this requires the mediator to be adapted, but the trust assumptions on the mediator are unchanged. The RP will verify the signature exactly as before.

Maintenance phase. As in Section V, the user can use the dealer to add and remove devices and decrease t. If the user wishes to add a new device as an authenticator, the dealer can generate a new share of k_a and, if it is consentful, a new share of k_c. If the user wishes to revoke a (non-consentful) authenticator, the dealer can re-distribute k_a and leave k_c unchanged. If the user wishes to revoke a consentful device, the dealer can generate fresh values for k_a and k_c, and then re-share them according to the decreased n. To decrease either t or t_c, the dealer can generate a new share of k_a or k_c and broadcast the new share to all authenticators or consentful authenticators respectively. As before, an increase of either t or t_c requires k_{priv} and the corresponding k_{pub} to be updated.

VII. Implementation

This section details the design considerations we explored in choosing how to implement the dealer. We also evaluated the feasibility of our design by implementing a proof of concept Symbolon authenticator and evaluating it against an *unmodified* WebAuthn RP both in terms of performance and compliance with the FIDO2/WebAuthn standards (see Table I).

As discussed in Section IV, the dealer must be *secure* since it holds the private key used to generate shares for Symbolon authenticators. However, unlike a traditional FIDO2 authenticator, the dealer is decoupled from the authentication phase and is only needed during less frequent operations like registration and changing the set of authenticators. As a result, the dealer is not required to be as *highly available* as the authenticator devices. This gives implementors options in how they secure the dealer when not in use. We consider two approaches that benefit different user profiles and requirements.

Dedicated local device. One possible realisation of the dealer is in the form of a physical device designed solely for this purpose. With this approach, the dealer can be kept offline in a secure location when not needed to minimise exposure to remote and physical adversaries. A physical device also enables pairing with new devices over short range media. For instance, a device with I/O interfaces such as a smartphone, Raspberry Pi with a UI shield, or purpose built IoT device can give user feedback on the set of enrolled devices and supports more secure pairing modes with user confirmation.

For dealers without sufficient I/O interfaces to perform registration and device management on their own, an intermediary device may be needed. For example, a dealer implemented as a hardened USB token can be plugged into a laptop with networking capabilities. Since the dealer would then depend on the intermediary device, care must be taken to protect the communication paths between the dealer and authenticators. If the intermediary is not trustworthy, one option would be to leverage available hardware security capabilities like Trusted Execution Environments (TEE) in combination with any solution that allows the creation of a secure communication channel between I/O peripherals and the TEE [13]). This enables users to securely interact with the secure USB token through the I/O interfaces of the (potentially malicious) laptop. Another option would be to use trusted computing capabilities that allow the dealer to verify the intermediary before exposing its services. For example, a Trusted Platform Module in combination with secure boot and an attestation framework can report the current configuration to the dealer for verification.

If it is anticipated the dealer will be unavailable for an upcoming registration phase, the dealer could generate and distribute shares of a public key k_{pub} and a handle amongst the authenticators in advance. When the user then wants to register to an RP when the dealer is unavailable, t authenticators could collaboratively act as the dealer and register with the RP by constructing and signature on (c, k_{pub}, H_k), as in Figure 3.

Another alternative if the dealer is unavailable is to allow for a 'throwaway' dealer, whereby an ephemeral yet trusted entity plays the role of the dealer and, after completing the registration phase with the RP, either deletes the signing key (forfeiting the ability to change the set of authenticators without updating the public key) or encrypts the signing key under a public encryption key belonging to the permanent dealer before then deleting the signing key.

Cloud service. Alternatively, users may desire a dealer with higher availability, which could be achieved by implementing the dealer as an online service. A cloud service could be realised in several ways. At the most basic level, the dealer's secrets could be stored securely in an online hardware security module (HSM) and only migrated temporarily to a trusted device when needed. A more convenient option would be to host the dealer's logic in a cloud as well. Tools like cloud-based HSMs that implement dealer logic or TEE-enabled Infrastructure as a Service (IaaS) solutions [1] eliminate the need to trust cloud providers or the underlying hardware and software where their workloads are executed. One inherent limitation of using a cloud service as a dealer is that users must authenticate to the cloud service. Users may opt to use layered

TABLE I: Comparison of registration and authentication duration for both implementations of standard FIDO2 and Symbolon authenticators. Timings are mean values in *ms* with 95% confidence intervals for a sample size of 100.

Protocol phase		**FIDO2**	**Symbolon** ($t = 2, n = 3$)	**Symbolon** ($t = 3, n = 4$)
Registration	(i) FIDO2 client	9.6 ± 1.0	11.8 ± 0.9	9.9 ± 1.8
	(ii) Public key selection	14.2 ± 0.4	15.6 ± 0.4	14.8 ± 0.9
	Total client side	23.8 ± 0.6	27.3 ± 0.6	24.7 ± 1.0
Authentication	(i) FIDO2 client	10.0 ± 0.5	10.1 ± 2.6	10.8 ± 1.2
	(ii) Signature generation	6.6 ± 0.2	28.1 ± 1.3	30.1 ± 0.8
	(iii) Mediator		19.3 ± 0.5	18.8 ± 0.8
	(iv) Authenticator		7.7 ± 0.3	9.2 ± 0.4
	Total client side	16.6 ± 0.4	38.2 ± 1.4	40.9 ± 0.8

MFA authentication schemes even if they are inconvenient as this is a less frequent, but sensitive task.

Recovery from loss. A dedicated dealer device may be lost, and a cloud service may become inaccessible (e.g., if the user loses their cloud credentials). If this happens, the user has to assert their identity to the RP using other means (e.g., calling the help desk), to revoke their dealer and enrol a new one.

VIII. Analysis

Here, we evaluate the properties offered by Symbolon, referencing the requirements identified in Section IV-B. Next, we use ProVerif to formally verify the security properties our threshold approach provides, then evaluate our system in an evaluation framework.

Threshold-based security (Sec1). Symbolon ensures users can authenticate to RPs if and only if they have (at least) t devices and give explicit consent on (at least) one of these devices. We prove this using ProVerif in Section VIII-A.

Authenticator privacy (Pri1). In Symbolon, RPs only need to store a single public key for each user account. Hence, RPs cannot distinguish between Symbolon and other public-key-based cryptographic protocols such as FIDO2. Another advantage of Symbolon is that it is fully implemented on the client side, which allows for concealing its underlying parameters from RPs. This includes, for example, the number of authenticators the user employs, the authenticators' capabilities and how often devices are added to or revoked from the list of authenticators. All this information can reveal sensitive information, which could allow the linkability of a user's activity between RPs. In addition, RPs cannot distinguish between sets of authenticators employed by the user to authenticate at different times. This is particularly relevant when the set of authenticators can disclose sensitive information such as their location at a given time.

Flexibility (Pra1). Symbolon allows users to login to RPs from a client device using any set of t authenticators, making the scheme resilient to attacks while providing flexibility to users. This is a desirable feature since users can specify different threshold policies depending on the RP they authenticate to, and can still login to RPs even if they forget or lose $n - t$ of their authenticators.

User control (Pra2). The process by which devices are added to or revoked from the list of authenticators is done locally without requiring involvement of RPs. This is also true for the selection of consentful authenticator(s) on which the user performs an action to consent to authenticating.

Authenticator set flexibility (Pra3). In Symbolon, adding or removing authenticators does *not* require users to update the public key stored by RPs. This feature is particularly useful when an authenticator is used for authenticating to several RPs, since it allows revocation without requiring the user to login into each RP and provide each with a new public key.

Compatible with FIDO2/WebAuthn (Pra4). Since Symbolon works by allowing a set of authenticators to perform the role of a FIDO2/WebAuthn authenticator, it is compatible with browsers and RPs built for authentication using FIDO2/WebAuthn.

The FIDO2/WebAuthn registration and authentication flows are identical from the server and browser perspective, as the signatures are computed by a threshold of authenticators and combined using threshold cryptography on the client side. We demonstrated this in our implementation (Section VII), where the mediator process emulates a USB device acting as a FIDO2/WebAuthn authenticator.

A. ProVerif analysis

ProVerif is a tool which allows one to model a cryptographic protocol and check its security properties. Protocol participants are defined in a process language which models input and output on channels, and computations on data. Cryptographic operations are defined using an equational theory, which is expressive enough to allow one to define threshold-based cryptography of the kind that Symbolon requires. We verify our core security property Sec1. Here, we briefly outline our method and results; more details are given in Appendix C in the full version of our paper [4].

Generating model instances. Symbolon is parameterised by four parameters: n (the number of authenticators), n_c (the number of authenticators capable of recording the user's consent), t (the threshold number of authenticators required to authenticate, and t_c (the number of those that are required to have obtained the user's consent). Unfortunately, the ProVerif language can only model *particular instances* of Symbolon, for example the instance $(n, n_c, t, t_c) = (5, 2, 3, 1)$.

To overcome this limitation of ProVerif, we wrote a Python script to generate all the models for n ranging from 2 to 8. Models must satisfy $n \geq n_c$, $t \geq t_c$, $n > t$ and $n_c \geq t_c$; the number of such models for each value of n is as follows:

n	2	3	4	5	6	7	8
no. of models	2	8	20	40	70	112	168

Note that we do not need to prove the properties for *all* n; we focus on the values of n that are reasonable to choose.

Modeling details. We declare a process Dealer, which creates the secret key and derives from it the corresponding public key, as well as n shares $k_1, \ldots, k_n$ and n_c 'consentful' key shares $ck_1, \ldots, ck_{n_c}$. It distributes k_i to processes representing the signers ($1 \leq i \leq n$), and additionally distributes ck_j to the consentful signers ($0 \leq j \leq n_c$). Authenticators sign the challenge from the RP using their share k_i, and their key ck_j if they have one. The threshold signatures are encoded as follows. Each device constructs its signature using its share(s), and then the function "combine" aggregates the partial signatures into the complete one. This complete signature is deemed valid (i.e., will be successfully verified by the RP) if the combined signature has t regular signature shares and t_c consentful ones, i.e., if one of the following "reduction rules" results in true:

reduc verify(pubkey, m,
combine(sign(k_{i_1}, m), …, sign(k_{i_t}, m), csign(ck_{j_1}, m), …, csign($k_{j_{t_c}}, m$)) = true

where the indices $i_1, \ldots, i_t$ are all distinct and $j_1, \ldots, j_{t_c}$ are all distinct, and the key shares k_i and ck_j are derived from a dealer master key. This means the arity of the combine function, and the number of reduction rules of this form, depend on the values of n, n_c, t, t_c. This is handled by our Python script that generates all the models.

We then use ProVerif to check that the RP will accept only signatures that have been made by a valid cohort of authenticators. This is achieved by the process RelyingParty declaring an event "authorised(c)" if it has accepted a signature on the challenge c, and Signer processes declaring events "signed(c)" and "csigned(c)" representing their ordinary and consentful signatures respectively. Then the query to verify at least t signers have signed is:

$$\text{authorised}(c) \Rightarrow \bigvee_{\substack{I \subseteq \{1,\ldots,n\} \\ |I|=t}} \bigwedge_{i \in I} \text{signed}(k_i, c)$$

and the query to verify t_c of them are consentful is similar.

Additional checks. We want to verify that, within a particular model for some n, if the number of authenticators is increased or decreased (because new devices are enrolled or removed), the authentication property still holds. To encode this in ProVerif, we let the attacker choose if and when each of the n signing keys are enabled. The attacker can enable or disable each of them at will, and thus fully control adding and removing signers (up to the limit of n signers in total).

Generating the models and running ProVerif. We generated all 420 ProVerif models for the values $2 \leq n \leq 8$ mentioned in the table above. Some of the model text files are more than 1MB, demonstrating the power of our method (such a model would be impossible to write by hand). We ran ProVerif 2.00 on all 420 models up to $n = 8$, and all the properties were successfully verified. Small models are verified in seconds; some larger ones took more than 30 hours. Our Python program for generating ProVerif code, and some examples of the generated ProVerif, are available at [4].

Conclusions. We verified the Sec1 property from Section IV-B for all reasonably-sized models (up to $n = 8$).

IX. Related work

Attacks against hardware tokens. Recently, multiple side channel attacks have been proposed that allow attackers with physical access to hardware tokens to efficiently extract their keys or bypass local authentication. For example, Oswald et al. demonstrate a practical, non-invasive side channel attack on Yubikey2 which extracts the AES key in one hour of access to the device [26]; Nemec et al. devised an attack to retrieve the private RSA key in Yubikey 4 devices [24]; Boneh et al. use hardware faults to break RSA signatures [8]; and Yen et al. demonstrate a fault attack on modular exponentiation [34]. These attacks show it is difficult to protect hardware tokens executing cryptographic functions from attackers with physical access. In this work, we advocate for the use of threshold cryptography across multiple authenticators to counter inherent security risks that emerge when hardware tokens utilise cryptographic keys. This is an advantage of threshold cryptography highlighted in the NIST standardisation process [9].

Existing solutions. *Google's Titan Security Keys (TSK)* are an authentication factor that protects users against phishing and MiTM attacks. They rely on public-key cryptography and contain a hardware chip resistant to physical attacks [21]. TSKs are compatible with browsers and RPs that support FIDO and unfortunately, they suffer from the same limitations as FIDO in terms of availability and security (see Section I).

Pico, presented by Stajano et al.'s, is a dedicated hardware token whose initial goal was to eliminate passwords through public-key cryptography [30]. The authors propose to keep the Pico 'locked' when the user is not present by encrypting its memory with a key that is secret shared among a set of other devices worn by the user (called Picosiblings) [32]. Only if a certain number of Picosiblings are nearby can Pico reconstruct the key and decrypt its memory. However, the authors decided to avoid public-key cryptography in their final version of Pico due to deployability issues [31], which made Pico a password manager implemented in a hardware token, and therefore does not replace passwords. Pico is still lacking a system specification and a full implemented prototype. Symbolon, like the original Pico idea, relies on secret sharing, but is based solely on public-key cryptography and does not require the client device to store or reconstruct any secrets.

n-Auth, introduced by Peeters et al., is a mobile authentication solution based on public-key cryptography [27]. To mitigate attacks on a stolen n-Auth device, the keys are encrypted on the device using a 4-digit PIN known to the user. The user's PIN is verified online using zero-knowledge techniques by a server that rate limits PIN entries, mitigating brute-force attacks and avoiding the server learning the PIN. If PIN verification is successful, the server sends some information

so the n-Auth device can compute short-lived symmetric keys to decrypt the private keys stored. Unlike Symbolon, n-Auth does not offer any mechanism to recover from loss and is only partially resilient to theft, due to learning (via shoulder surfing or by using malware) or guessing a 4-digit PIN is not out of reach of many adversaries, despite the server's rate limiting.

Shatter leverages threshold cryptography to distribute private signing keys of their apps amongst a set of their devices [5]. Shatter must be installed on all the user's devices and can be run without modifying the apps. When a cryptographic operation must be performed, Shatter queries each of the user's devices to obtain their contributions to the operation; devices are configured either to conduct the operation automatically or to first ask the user to consent.

Shatter takes a different architecture approach to Symbolon and requires the signing key to be reconstructed on one of the authenticators for maintenance operations (i.e., adding or removing authenticators). Because this reconstruction can happen on any of the authenticators, every authenticator must be trusted to reconstruct, re-distribute and then delete the key. This is a large trusted computing base, especially as the authenticators can be generic user devices and may have large amounts of software on them. If the authenticator reconstructing the key is infected with malware, the malware could learn the key, despite compromising only a single authenticator. In contrast, Symbolon takes a centralised approach to maintenance by introducing the dealer, which stores the key rather than having authenticators reconstruct the key. This significantly reduces the trusted computing base compared to Shatter, which we think unrealistically large, and protects our system so malware on fewer than t of the user's authenticators cannot learn the signing key, as is possible in Shatter.

Shatter's solution to user consent is also different to ours. In Shatter, each device is individually configured to either contribute to the signing operation automatically, or only after consent is received. This may result in the user being required to consent on every device contributing to the authentication, a potentially burdensome activity. In contrast, Symbolon defines a consent threshold t_c, in addition to t, and distributes 'consentful shares' (shares of k_c, to be used only when user consent is registered). This key distribution cryptographically enforces that t primary shares and t_c consentful shares are required in order for a valid signature to be produced. Furthermore, this distribution enables the user to choose which t_c devices are most convenient for them to consent on per authentication session. Thus, we extend the flexibility offered by threshold cryptography to enforcing user consent, rather than defining it per-device as Shatter does.

X. Conclusions

We presented Symbolon, a multi-device-based user authentication solution which requires a threshold number of users' authenticators to be present, and for the user to provide consent on (at least) one of their authenticators, in order for the authentication to be successful. Our solution is fully implemented on the client side and so is compatible with relying parties that already employ public key based challenge response user authentication systems, such as FIDO2/WebAuthn. After presenting our solution, we analysed its security, privacy and deployability. We found Symbolon offers enhanced security properties over single factor FIDO: it inherits many security properties from FIDO, but also it is resilient to theft of up to $t-1$ devices. Our solution also provides some other benefits as authenticators can be added and removed by leveraging the dealer (a trusted entity used for management of the system that is owned and controlled by the user) without the user having to contact each relying party on an account-by-account basis. Symbolon offers multiple privacy benefits to the user and lets them choose, implement and enforce their own policy without having to rely on the relying party. Finally, we implemented our solution as an extension to FIDO/WebAuthn, and presented measurements to demonstrate its feasibility. In our future work, we will explore the use of shared devices like printers, networking equipment, and common infrastructure that may provide additional context during authentication.

Acknowledgments

Mark Ryan gratefully acknowledges his appointment as *HP Research Chair* generously supported by HP Labs. We also gratefully acknowledge financial support from EPSRC under grants EP/V000454/1 (CAP-TEE: Capability Architectures for Trusted Execution); EP/S030867/1 (SIPP - Secure IoT Processor Platform with Remote Attestation); and EP/R012598/1 (User-controlled hardware security anchors: evaluation and designs).

References

[1] "Introducing Azure confidential computing," https://azure.microsoft.com/en-us/blog/introducing-azure-confidential-computing/.

[2] A. Adams and M. Sasse, "Users are not the enemy," *Communications of the ACM*, vol. 42, no. 12, pp. 40–46, 1999.

[3] F. Alliance, "Client to Authenticator Protocol (CTAP)." [Online]. Available: https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html

[4] Anon., "Symbolon: Enabling Flexible Multi-device-based User Authentication," 2021. [Online]. Available: https://www.dropbox.com/sh/rh0n9qm1gnf7k0y/AAAAcx8_0Ayw9YGEdYnQhEmda?dl=0

[5] E. Atwater and U. Hengartner, "Shatter: Using Threshold Cryptography to Protect Single Users with Multiple Devices," in *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2016, pp. 91–102.

[6] G. Blakley, "Safeguarding cryptographic keys," in *International Workshop on Managing Requirements Knowledge (MARK)*, 1979, pp. 313–318.

[7] Bluetooth S.I.G, "Bluetooth SIG. Bluetooth Core Specification 5.2," [Online; accessed 20-March-2020]. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification/

[8] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1997, pp. 37–51.

[9] L. Brandão, M. Davidson, and A. Vassilev, "NIST roadmap toward criteria for threshold schemes for cryptographic primitives," National Institute of Standards and Technology, Tech. Rep., 2020.

[10] C. Castelluccia and P. Mutaf, "Shake them up! a movement-based pairing protocol for cpu-constrained devices," in *International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2005, pp. 51–64.

[11] I. Damgård and M. Koprowski, "Practical threshold RSA signatures without a trusted dealer," in *International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, 2001, pp. 152–165.

[12] D. Dolev and A. Yao, "On the Security of Public Key Protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 2006.

[13] S. Eskandarian, J. Cogan, S. Birnbaum, P. C. W. Brandon, D. Franke, F. Fraser, G. Garcia, E. Gong, H. T. Nguyen, T. K. Sethi, V. Subbiah, M. Backes, G. Pellegrino, and D. Boneh, "Fidelius: Protecting User Secrets from Compromised Browsers," in *IEEE Symposium on Security and Privacy (IEEE S&P)*, 2019, pp. 264–280.

[14] FIDO Alliance, "FIDO Universal 2nd Factor," [Online; accessed 20-March-2020]. [Online]. Available: https://fidoalliance.org/

[15] C. Gehrmann, C. Mitchell, and K. Nyberg, "Manual authentication for wireless devices," *RSA Cryptobytes*, vol. 7, no. 1, pp. 29–37, 2004.

[16] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ECDSA with fast trustless setup," in *ACM Conference on Computer and Communications Security (CCS)*, 2018, pp. 1179–1194.

[17] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security," in *International Conference on Applied Cryptography and Network Security (ACNS)*, 2016, pp. 156–174.

[18] P. Grassi, J. Fenton, E. Newton, R. Perlner, A. Regenscheid, W. Burr, J. Richer, N. Lefkovitz, J. Danker, Y. Choong *et al.*, "Nist special publication 800-63b: digital identity guidelines," *Enrollment and Identity Proofing Requirements. url: https://pages. nist. gov/800-63-3/sp800-63a. html*, 2017.

[19] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Annual International Cryptology Conference (CRYPTO)*, 1995, pp. 339–352.

[20] M. Ito, A. Saito, and T. Nishizeki, "Secret sharing scheme realizing general access structure," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 72, no. 9, pp. 56–64, 1989.

[21] J. Lang, A. Czeskis, D. Balfanz, M. Schilder, and S. Srinivas, "Security keys: Practical cryptographic second factors for the modern web," in *International Conference on Financial Cryptography and Data Security (FC*, 2016, pp. 422–440.

[22] S. G. Lyastani, M. Schilling, M. Neumayr, M. Backes, and S. Bugiel, "Is FIDO2 the Kingslayer of User Authentication? A Comparative Usability Study of FIDO2 Passwordless Authentication," in *IEEE Symposium on Security and Privacy (IEEE S&P)*, 2020.

[23] R. Morris and K. Thompson, "Password security: A case history," *Communications of the ACM*, vol. 22, no. 11, pp. 594–597, 1979.

[24] M. Nemec, M. Sys, P. Svenda, D. Klinec, and V. Matyas, "The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli," in *ACM Conference on Computer and Communications Security (CCS)*, 2017, pp. 1631–1648.

[25] M. Nojoumian, D. Stinson, and M. Grainger, "Unconditionally secure social secret sharing scheme," vol. 4, no. 4, 2010, pp. 202–211.

[26] D. Oswald, B. Richter, and C. Paar, "Side-channel attacks on the Yubikey 2 one-time password generator," in *International Workshop on Recent Advances in Intrusion Detection (RAID)*, 2013, pp. 204–222.

[27] R. Peeters, J. Hermans, P. Maene, K. Grenman, K. Halunen, and J. Häikiö, "N-Auth: Mobile Authentication Done Right," in *Annual Computer Security Applications Conference (ACSAC)*, 2017, pp. 1–15.

[28] A. Shamir, "How to share a secret," vol. 22, no. 11, 1979, pp. 612–613.

[29] V. Shoup, "Practical threshold signatures," in *International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, 2000, pp. 207–220.

[30] F. Stajano, "Pico: No More Passwords!" in *International Workshop on Security Protocols*, 2011, pp. 49–81.

[31] F. Stajano, B. Christianson, M. Lomas, G. Jenkinson, J. Payne, M. Spencer, and Q. Stafford-Fraser, "Pico Without Public Keys," in *International Workshop on Security Protocols*, 2015, pp. 195–211.

[32] O. Stannard and F. Stajano, "Am I in Good Company? A Privacy-Protecting Protocol for Cooperating Ubiquitous Computing Devices," in *Security Protocols XX*, 2012, pp. 223–230.

[33] D. Stinson and R. Strobl, "Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates," in *Australasian Conference on Information Security and Privacy (ACISP)*, 2001, pp. 417–434.

[34] Y. Sung-Ming, S. Kim, S. Lim, and S. Moon, "A countermeasure against one physical cryptanalysis may benefit another attack," in *International Conference on Information Security and Cryptology*. Springer, 2001, pp. 414–427.

[35] W3C, "Web Authentication: An API for accessing Public Key Credentials Level 1," March 2019, [Online; accessed 01-May-2020]. [Online]. Available: https://www.w3.org/TR/webauthn/