Threshold Scheme Implementation for Physical Security Keys

Keawe Mann, Jess Ding and Ashhad Alam

May 15, 2024

Abstract

As we move into a more digital world, the commercial use of physical security keys has been rising. However, there is still a barrier to entry for the regular customer because of the difficulty in recovering access in case the key is lost. In this paper, we propose a proof-of-concept implementation of threshold cryptography to support multi-party keys. This will reduce some of the risk surrounding physical security keys and make them more accessible for personal use.

1 Introduction

As we move into a more digital world, creating strong passwords is even more important than before. However, the number of people being affected by data breaches has been increasing at an alarming rate. This means that relying on strong passwords is not enough, and we need other ways to keep our data secure. One of the ways to do so is Two-Factor Authentication (2FA) or its more general counterpart, Multi-Factor Authentication (MFA).

2 Motivation and Related Work

Two-factor authentication adds an additional layer of security to verify that only authorized users can gain access to the data requested. Rather than immediately gaining access after entering the correct username and password, users are prompted to enter some more information or take some additional action.

Many major account providers (Google, Meta, Apple) for various email, social networking, and other identity-based services are now transitioning their account holders to use some form of Multi-Factor Authentication (MFA), whether it be through a separate authenticator service like Duo, Authy, or Yubikey, or through OTP-based methods. Unlike Google's Youtube verification or Apple iCloud's device verification, such separate authenticator services are third-parties that do not require internal knowledge of how the account provider's services work, which is why users who want additional layers of security on their accounts tend to use these third party authentication services. Among these third party authenticator services, app solutions like Duo and Authy provide relative convenience for logging in, but physical security keys like Yubikey are again used more by security buffs, since their offline nature again reduces network-based security risks. Of course, the added security of physical authenticators comes at the tradeoff of convenience, since users must always have that physical key with them to unlock their online accounts.

Current best practices for using physical authenticators places a heavy burden on users. Users are encouraged to preemptively buy and set up up duplicate keys for their accounts, ensuring they will retain access to these accounts, even after losing their primary key. Some users go so far as to store these spare keys in safes or deposit boxes. Without a backdoor into the accounts secured by a physical key, having only one physical key requires the user to relinquish a large amount of convenience for a comparatively small amount of added security. This security catch-22 is why physical security keys are mostly only used for corporate accounts, where the IT department usually has the account backdoor, or for security buffs who invest into multiple physical keys to hedge against loss (famously, for cold bitcoin wallets where online security is only as good as offline physical security).

In the realm of Multi-Party Authorization (MPA), processes exist for protecting telecommunications, data, and other services of national interest. Such processes can be implemented technically, though many other processes also hinge heavily on procedural implementations, as in the case of the Soviet submarine commander who helped avert international nuclear war during the Cuban missile crisis. In any case, current such MPA processes have too much overhead to be practical for regular email account security, though the concept warrants exploration for a mini-fied solution.

More academically, threshold schemes with a centralized trusted entity and threshold cryptography with entirely distributed security have applications in cloud computing and low-energy and low-latency situations [3]. There are various threshold cryptography schemes, including one by RSA Security [2] [1]. In such a system, there is a public key, while the private key is shared between a group, in such a way that some subset of the group can collaborate to figure out the private key. An adversary, on the other hand, will need to gain access to a non-trivial number of group members to be able to learn anything about the private key.

Given the convenience-security tradeoff of available MFA implementations, especially physical security keys, we propose a proof-of-concept implementation of a threshold scheme for account recovery. Allowing trusted family and friends to together provide a backdoor into your account only when a user has reported their physical authentication keys as lost will lower the convenience cost for ordinary users to increase their account security. More concretely, we propose a proof-of-concept scheme where a single security key can unlock a demo account (normal use case scenario), or where several people working together (at least t out of a group of size n), with simulated security keys, can also unlock the account.

Most existing account recovery methods necessarily reduce security in some way. Existing options include an email sent to a designated account or a text sent to a designated phone. Introducing these separate systems and accounts associated with account recovery significantly increases the attack surface. Merely requesting security changes/password resets and compromising a user's email could be enough to break into the user's account.

With our scheme, the recovery method does not reduce the security of the overall system. Several actors must work together to unlock the account. These same actors also cannot collaborate to open the account for themselves—only the original user will have access. Moreover, each of the actors' shares, which must be combined to provide access to the account (as described in later sections), are guarded by security keys. This ensures at least one security key is involved any access to the account, even during the recovery process.

3 Technical Background

Our implementation relies directly on Shamir's secret sharing scheme as presented in class: in a t of n scheme, generate a random polynomial of degree $t-1$ if GF$[p]$, $p > n$ where $f(0) = m$ and share $i \in [1, n]$ is $f(i)$. Reconstruction involves solving a system of t equations in t variables, and correctness and security follow from class.

The field of threshold cryptography is a natural extension of Shamir's secret sharing, but for wider deployments like distributed computing or private matrix multiplication.

4 Implementation

The implementation details for the scheme will focus on the following:

- The standard login process
- Assigning trusted users
- Distributing shares
- Recovering the user's account in case the key is lost

4.1 Standard Login Process

For any user in our scheme, the standard login process comprises of a two-factor authentication process. The user will have to enter their password then use their security key to gain access to their account. An incorrect password, a non-existent security key, or a security key registered to a different user will not allow access to the user's account.

Even though the use of the security key makes the login process more secure, the user is still encouraged to choose a strong password because of the account recovery process outlined in Section 4.4.

Our implementation uses the Web Authentication (WebAuthn) standard, which is widely used in industry. WebAuthn interfaces with security keys (or passkeys, an increasingly popular alternative to physical security keys) to allow for 2FA. This standard also allows for passwordless (single factor) authentication, where only the security key and a username are required to log in. Our implementation of WebAuthn uses Python on the backend (running on the host), and JavaScript for the front end (running on the client device). We also use the Python Flask library as needed to run our website.

WebAuthn has two procedures, one for registration and one for login. For both registration and login, the client requests a challenge from the host. This challenge is randomly generated by the host and guards against replay attacks. A new challenge is generated for each login and registration.

For registration, once a challenge has been received from the host, the client platform prompts the user to set up a security key or passkey. Next, a credential is produced with the help of the security key. This credential contains a unique ID, the challenge, a public key, and several other attributes. Importantly, the security key stores the private key counterpart of the aforementioned public key. The public key will be used by the host to verify signatures sent with future login requests. The credential is then sent to the host to verify. The host saves certain information from this credential, such as the ID and public key, for any future login requests. The challenge contained in the credential should match the most recent challenge the host sent out for the given user. If everything checks out, registration has succeeded.

For login, as previously mentioned, the client again requests a challenge from the host. However, unlike with registration, the client also requests the unique ID that the host has been saving from registration. Back on the client, the unique ID, plus the challenge and a few other fields, is provided as input to create a login request. The user's security key uses the private key (if one exists) corresponding with the provided ID to create a signature. This signature is sent along with several other data attributes to the host. If the host is able to verify the signature and the correct challenge has been sent back, login succeeds.

WebAuthn allows for severThe mentioned public and private key are elliptic curv

4.2 Assigning Trusted Users

Once a user has created their account and logged in with their credentials, they can select any number of existing users to be their trusted users. In addition, the user will also select the threshold for how many users they need to approve any recovery request initialized in the future.

The user needs to be careful in assigning these trusted users, as untrustworthy users can pose a security risk, as outlined in 5.1.2. While our implementation does not take these steps, for a secure system, standard security procedures on storing user data and passwords should be followed. For example, only hashes of the passwords (with salt) should be stored and user data should all be encrypted.

4.3 Distributing Shares

The server will now generate a random password, P_w. It will then generate a $t-1$-degree polynomial (where t is the the threshold selected in Section 4.2) according to Shamir's secret sharing protocol. Next, the server will generate n shares (where n is the number of users selected in Section 4.2) and send these shares to each of the trusted users' accounts. The trusted user's accounts must also be secured with security keys.

In our implementation, shares are simply stored in each trusted user's account. In a real implementation, these shares should be stored encrypted. Perhaps with some additional hardware support, shares could be stored on trusted users' security keys, meaning no shares would have to be stored by the host server for any extended portion of time, further improving security. We used Python, along with the SageMath library, to implement Shamir's secret sharing protocol.

If user data is encrypted on the server, this data should be encrypted twice. Of course, this will double the amount of space the user's data takes up. One set of encrypted data will be encrypted in such a fashion that the user can enable decryption through the standard login process (i.e. when they still have their security key). To enable recovery of the user's data once this original security key has been lost, all user data should also be encrypted with P_w. This data could then be decrypted after we've reconstructed P_w at the end of the recovery process.

4.4 Account Recovery

The primary user will need to request an account recovery if either of the following happens:

- They lose their security key
- They forget their password

If the user loses their security key, they are still able to access their account in a limited capacity and request account recovery. They will not be given full access, or be able to make changes to their account in such a case. If the user forgot their password, they can request an account recovery using their security key in a similar manner. The figure below illustrates how the server might respond to different scenarios:

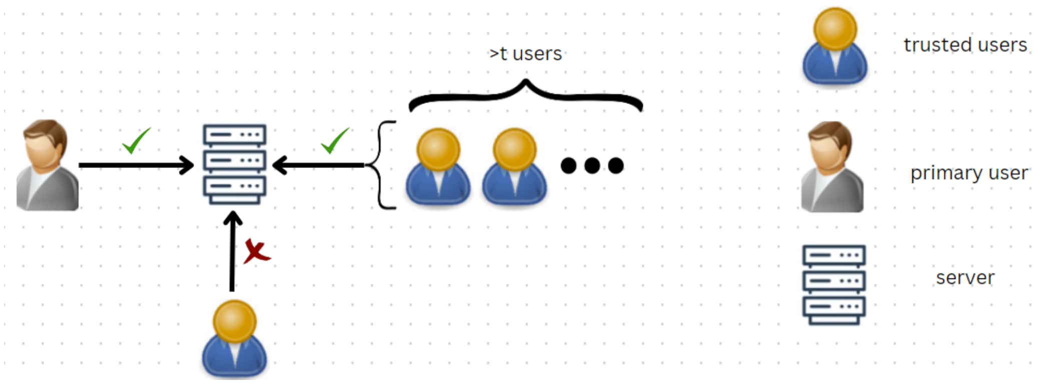

Figure 1: How trusted users can help recover the primary user's account

Once the user submits such a request, all of the trusted users will get a notification to send their shares. The server will wait until it gets at least t requests to reconstruct P_w generated in Section 4.3. Any less than t shares, and the server will be unsuccessful in recovering the secret password.

Once the password has been reconstructed, the primary user will now be given temporary access to their account. They will now be able to register a new security key, as well as reset their password. Once this has been done, they will be prompted to login again, and gain full access to their account.

5 Analysis

5.1 Security

Any adversary choosing to gain access to the system has two main target areas:

- The login process
- The recovery process

5.1.1 The Login Process

The login process involves the user's password and the physical security key. Even in the case that a user's password is insecure, or exposed (if, for instance, the user uses the same password everywhere and was the victim of a data breach), any attacker will still need to gain access to the physical security key to be able to access the account. Guessing the signature generated by the physical security key is not an easy task. Accordingly, the scheme is secure against attacks on the user's login process.

However, just knowing the user's password will still allow an adversary to mark the security key as lost. As a result, we need to be careful about verifying if it's actually the primary user who initiated this request, before letting the trusted users send shares back to the server. This is not covered as part of this project, and outlined as future work in Section 6. It is worth noting that even this would still not compromise the security of the account unless enough trusted users all send their shares.

5.1.2 The Recovery Process

The recovery process involves the trusted users sending their shares back to the server to reconstruct the random password. The security for this relies on Shamir Secret Sharing being secure. Since we need at least t shares to recover the polynomial used to hide the secret, any adversarial method will rely on corruption of at least t users. Even the corruption of $t - 1$ users will not reveal any information about the polynomial chosen, and hence, the secret message.

Since the trusted users' accounts are also secured using Two Factor Authorization with their password and security key, any attacks involving gaining access to these accounts are likely to be unsuccessful, as outlined in Section 5.1.1.

On the other hand, there is a risk of the trusted users colluding with each other to learn the secret. Security against this relies on making sure the primary user has a secure password (since this is needed to initialize the recovery process), proper loss report verification, as well as making sure the user picks their "trusted users" carefully. This is, unfortunately, one of the weaker points of the scheme and outlined as future work in Section 6.

5.1.3 Single Point of Failure

One big caveat to using this scheme is that it results in a singular point of failure. The secret password, P_w is generated at a single point before being split into shares. In addition, the shares are collected at a single point to reconstruct the secret. So, the scheme is very vulnerable to attacks on these points. If either of these servers crash, then the system cannot generate/reconstruct the secret password for recovery. This is a security vulnerability of this system.

5.2 Correctness

Shamir Secret Sharing generates $n(> t)$ shares of a polynomial of at most degree $t - 1$. Using t of these shares, we can guarantee reconstruction of this polynomial, and hence, find the secret. As a result, if we have at least t users collaborating to recover the secret, we can generate the passkey that was used to generate these shares. As a result, the scheme is correct.

6 Future Work

6.1 Weighted Sharing for Trusted Users

The way the scheme is designed allows easy expansion to allow tiered trusted users. In any structured organization, some users will have more powers than others. A simple way to implement such a feature would be to distribute shares to these users according to the power they should have. Consider the following example with two tiers of users:

We want to establish a scheme where each of the following options should be enough to recover the account:

- 2 **Tier I** users
- 6 **Tier II** users
- 1 **Tier I** user and 3 **Tier II** users

Now, instead of giving one threshold to each user, we can set our threshold (t) to 6. Each **Tier I** user gets 3 shares each and each **Tier II** user gets one share, which makes recovering the account possible in each of the above scenarios. This example can be expanded to include an arbitrary number of tiers, and the threshold can be chosen accordingly.

6.2 Dealing with Security Concerns

While this scheme makes recovering account access more secure, it does come at a cost. Shamir's secret-sharing scheme inherently doesn't account for any malicious actors who may submit incorrect shares, and the threshold scheme has a single point of failure at the trusted central entity that generates and reconstructs shares. Our proof-of-concept implementation also entirely reconstructs the exact password.

Beyond our proof-of-concept implementation reconstructing the exact user password, it should be easy to generate a different temporary passphrase to momentarily allow regained access when the user reports physical key loss, while not compromising on safety. We note that future work on a truly deployable solution should also include share verification, as well as look to improve having a single centralized entity for share generation and reconstruction. An implementation for actually verifying that the user themself is reporting their key as lost should also not be discounted as trivial in a truly deployable solution.

6.3 Expansion to Other Areas

Our implementation focuses on a scheme for account recovery using threshold cryptography for security keys. This can be expanded to other areas, where the goal is more than just account recovery. For instance, in the case of wills or digital currency wallets, we can use a similar method to the one proposed in this paper to gain access to the account. Obviously, the security of such a scheme is even more important in these use cases, and more work needs to be done to make the scheme secure before deploying it in these areas.

6.4 Cost-Benefit Analysis

Finally, we note that one detail remains: ascertaining if the market exists. That is, a truly deployable solution will also need to determine whether a user who cares about security enough to actually use a physical authenticator will also trust enough people to use a threshold implementation for key recovery. Given that we are not business majors, we shall leave this detail to the reader.

7 Team Contributions

Each team member worked on every portion of the project (proposal, implementation, presentation, and report).

References

[1] Ivan Damgård and Maciej Koprowski. Practical threshold rsa signatures without a trusted dealer. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 152–165. Springer, 2001.

[2] Hanna Lee, Hao Shen, and Brian Wheatman. Implementation and discussion of threshold rsa. 2016.

[3] Guoyan Zhang and Jing Qin. Lattice-based threshold cryptography and its applications in distributed cloud computing. *International Journal of High Performance Computing and Networking*, 8(2):176–185, 2015.

[4] Threshold Cryptography, MPC, and MultiSigs: A Complete Overview