KeyClub and RandRec: Two New Social Key Recovery Schemes

Supra Research Team

September 27, 2024

Abstract

We describe two new social recovery schemes using similar approaches but in different settings. Our first setting considers a community of key holders, the members of which can be thought of as crypto-natives. In this setting, we design a social recovery scheme – which we call **KeyClub** – in that a key owner uses (a subset of) other community members as guardians, who do not need to store any *new* information apart from their own pre-generated secret key (possibly used for signing transactions etc.). Additionally they may use passwords (which is not stored anywhere) to enhance security. Our second setting considers the guardians to be any Web2 user, who may not have a secret storage capacity, and relies solely on passwords. However, given access to a VRF service (that holds a secret VRF key) they can still support a back-up and recovery procedures similar to the earlier scheme – we call this scheme **RandRec**. Both our schemes rely on a core primitive, which we define as bottom-up secret sharing (BUSS), that enables an arbitrary $(t+1)$-out-of-n secret sharing for any given shares through additional public information, which is necessary during reconstruction and is possibly stored reliably on a blockchain.

1 Introduction

Cryptography plays a crucial role in securing and authenticating data in the digital space by providing mathematically proven guarantees. However, virtually all such guarantees crucially rely on keeping the underlying secret key secure and available, both at the same time. In the blockchain space, due to its fundamental reliance of cryptography, creating secure and reliable (that is available when needed) wallet services, for storing keys, has garnered substantial attention [7, 9] in the past few years.

Storing secret keys in wallets *securely* and *reliably* turns out to be remarkably challenging. Among many a primary challenge is, unlike passwords, keys can not be reset easily. For example, if certain funds are "locked" with respect to a particular public-key, such that one needs to produce a signature using the corresponding secret-key, then those funds are lost forever if the key cannot be

recovered. It is estimated that USD 140 billions worth of BTC is unrecoverable due to lost secret keys [11]! Therefore, many existing wallets support a backup option, either via mnemonic pass phrases [2], which one may write down in a secure place, or splitting the key [1, 3] using simple secret sharing (such as Shamir's[10]) and storing the shares in different devices – each share must be secured with another authentication mechanism, such as passwords. But even for those solutions (also called cold [8] or hardware wallets) incidental memory erasure or losing passwords (or a combination)[1] may happen realistically invoking a loss of fund. In fact, in scenarios involving permanent disappearance, such as death or disability of the key-owner, similar loss of fund can take place.

Many of these issues are mitigated in social recovery solutions, which, as layed out by Vitalik Butterin [5], carry substantial benefits in terms of usability and reliability without compromising security. In a social recovery scheme, a *key-owner* uses parties from her social circle, also known as *guardians*, for backing up her secret key. A typical recovery access structure can be $(t+1)$-out-of-n threshold, where n guardians are used for backup and any $(t+1)$ of them are required to recover the key. This setting will be secure as long as at most t of the shares are captured, by collusion or otherwise. To reduce the possibility of collusion, we would require the guardians not to know each other's identities. Importantly, for the event of permanent disappearance of the key-owner, the legitimate nominee can coordinate with any $(t+1)$ guardians to recover the secret-key and thus inherit any asset locked with the key.

Using a simple $(t+1)$-out-n secret sharing for social recovery, however, incurs new issues, as we elaborate next: let sk_i is party P_i's secret key, which is a 256 bit string. A secret sharing of sk_i would generate n shares $\sigma_{i,1}, \sigma_{i,2}, \ldots \sigma_{i,n}$, where $\sigma_{i,j}$ belongs to guardian P_j – each $\sigma_{i,j}$ is also a 256 bit string. Not only it is hard to securely and reliably store the shares (that basically requires the same procedure used to store keys), it scales poorly – if a party P_j acts as a guardian for many parties – for each key-owner P_k, the guardian P_j needs to store a separate share $\sigma_{k,j}$.

We mitigate this by using a technique, that we call bottom-up secret sharing (BUSS), which allows arbitrary $(t+1)$-out-of-n secret sharing even for any given shares, as long as they are randomly distributed (otherwise guessing the key becomes easier).[2] So, the problem now boils down to constructing random shares with as minimal information as possible.

KeyClub In our first setting we consider a community of key-holders, each of whom is a crypto-native or crypto-enthusiast and holds a secret/signing key, possibly used for signing transactions. There is a corresponding public/wallet key,

[1] In [5], a real world example was given for a Bitcoin developer Stefan Thomas, who had three backups entities – an encrypted USB stick, a Dropbox account and a Virtualbox virtual machine. However, he accidentally erased two of them and forgot the password of the third, forever losing access to 7,000 BTC (worth $\$125,000$ at the time).

[2] Note that, as long as the shares come from high entropy distribution, the scheme works. For simplicity, in this document, we stick to random shares.

given which the secret key is unpredictable. In this setting we design a simple social key recovery scheme: each key owner's key can be backed up with (a subset) of the rest of the community members. For recovery, any $(t+1)$ guardians must help correctly. The scheme supports every party to back up their respective keys with everyone else, without needing to store anything apart from their own secret key – this establishes an ecosystem of mutual dependence without additional overhead. This satisfies stronger security requirements against any *malicious* and *adaptive* corruption up to t parties. Even if all guardian's secrets are leaked, passwords can be used to ensure that the attacker still needs to execute an offline attack. We also emphasize that, both our backup and recovery protocols require only a single round trip interaction in a star network, with the key owner in the center, and guardians sending a single message, without requiring any synchronization among themselves – in fact the guardians do not require to know each other.

RandRec In the second setting the guardians are not assumed to be crypto-natives, and solely rely on passwords. In this case, we additionally rely on a VRF service, holding a VRF secret key. In particular, each guardian now interacts with the VRF service to derive a share – the pseudorandomness of which is guaranteed by the VRF property. Like above, the same security guarantees can be achieved against any *malicious* and *adaptive* corruption up to t parties. Additionally, even if the VRF server is malicious, it won't get any information about the guardian's passwords – this is ensured by using an input-oblivious VRF computation. Similar to above, also in this scheme the backup and recovery protocols communicate with the key-owner in a star-network fashion. Additionally, one more interaction is needed with the VRF server to derive the shares. Still, the guardians require no synchronization among themselves, and neither do they need to know each other's identities.

We emphasize that, though we describe separately, these two approaches maybe combined together. In particular, it is possible that some guardians are crypto-native and use their wallets while others are not crypto-native and rely instead on the VRF service.

2 Notation and Preliminaries

Notations We use $\mathbb{N}$ to denote a set of positive integers, and $[n]$ to denote the set $\{1, \ldots, n\}$ for any $n \in \mathbb{N}$. We denote the security parameter by $\lambda \in \mathbb{N}$. A set $X = \{x_1, \ldots, x_n\}$ is denoted as $x_{[n]}$ or $\{x_i\}_{i\in[n]}$. For any subset $S \subset [n]$, x_S or $\{x_i\}_{i\in S}$ denotes the subset of X containing all x_i's such that $i \in S$. A ordered tuple $(x_1, \ldots, x_n)$ is denoted by vector notation $\vec{x}_{[n]}$ or $(x_i)_{i\in[n]}$. Similarly for any subset S of $[n]$, $\vec{x}_S$ and $(x_i)_{i\in S}$ are defined.

Every algorithm takes security parameter λ as an input, even if not always mentioned explicitly; all definitions work for a sufficiently large choice of λ. A typical value for λ is 256. We use $y := D(x)$ to denote a deterministic

computation and $y := x$ for assignment. Randomized computations are denoted as $y \leftarrow R(x)$. The symbol $\perp$ denotes an undefined value, or invalidity.

Communication Model We assume that parties are connected by pairwise secure and authenticated channels, such as TLS. Furthermore, everyone has access to an immutable public bulletin board, where data can be stored reliably, so that they are available when needed. For example, we may assume that such bulletin board is realized by a smart-contract over a blockchain.

Threat Model We consider a threat model, where an attacker can corrupt upto t parties in the system, arbitrarily. Our protocols do not have an identifiable abort feature, where in case of abort, a corrupt party could have been identified. For the VRF based protocol, in addition to the standard corruption, the VRF service can be totally untrusted – each response from VRF can be verified, and due to input-obliviousness it does not know anything about guardian's input, that contain passwords. Furthermore, using passwords would partially protect the key in the worst case scenario, in that all guardian's secrets are compromised, forcing the attacker to brute force the passwords using off-line dictionary attack.

3 Syntax and Semantics

In this section we present the basic syntax of a social recovery scheme. Also we provide brief discussions about the semantics of security and correctness without presenting formal definitions.

We note that the social recovery schemes we consider here do not require the guardians to remember / store any additional information apart from their a priori secret state (possibly their own signing key), and a password. It suffices for the back-up procedure to produce only *public* back-up information pub, to be stored on the blockchain / public bulletin board reliably. Also for the basic schemes we assume that, for a secret key, the shares are fixed, such that if the same key is shared multiple times, the same shares (both private and public) would be generated – this means we may just consider the back protocol to be executed once. Later in Section 7 we discuss how to enable sharing of the same keys multiple times with different shares.

Syntax Consider that each P_i has an identity, denoted by integer i. For party P_i, there maybe a key-generation algorithm KeyGen_i, which produces a pair of keys $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}_i(1^\lambda)$, pk_i is published, and posted on the blockchain / bulletin board. An access structure contains all pairs of sets (B, R) and indicates whether parties in set R is allowed to recover the key that was backed up by parties in set B. We only consider a $(t+1)$-threshold access structure, which consists of all pairs (B, R) such that $|R| = (t+1)$ and $R \subseteq B \subset [N]$. In this setting, a social recovery scheme Π_{SKR} consists of a pair of

protocols $(\Pi_{\text{Back}}, \Pi_{\text{Rec}})$ executed in the following order: each party may initiate an instance of Π_{Back} once with a set of guardians B; finally once P_i has finished executing Π_{Back}, it can initiate Π_{Rec} arbitrarily many times with any subset $R \subseteq B$. The protocols have the following syntax:

- Π_{Back}: In this protocol, a key owner P_i who wishes to back up her secret key sk_i interacts with a set of guardians $\{P_j\}_{j \in B}$. Each guardian P_j interact with the owner, but doesn't interact with each other. The protocol concludes with a *public* back-up string pub_i, published on the blockchain.
- Π_{Rec}: If P_i wishes to recover sk_i using a recovery set $R \subseteq B$, she runs this protocol without any secret input with a set of guardians $\{P_j\}_{j \in R}$. In this procedure pub_i may be used by all parties. At the end of this protocol, the key-owner may receive a private input sk_i (or $\perp$ if unsuccessful).

Now we discuss the structural correctness and security requirements.

Structural Requirements We require that:

- The protocols Π_{Back} and Π_{Rec} are both star-network protocol, in that the owner is at the center of the star, and the guardians don't interact among themselves. It maybe possible that they interact with other external entities, for example, in RandRec the guardians interact with the VRF server(s).
- The guardians do not even need to know the identity of the other guardians. Only the key-owner knows the set B, and needs to remember this for recovery. This would reduce the possibility of collusion as discussed by Vitalik [5].

Correctness For correctness of the scheme we need that for any $i \in [N]$, any pair of back-up and recovery sets (B, R) that is within the access structure (for threshold we require $|R = (t+1)|$), then for any owner P_i, who backs up a secret key sk_i generated as $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}_i(1^\lambda)$ using B by running Π_{Back} and recovers by executing Π_{Rec} with R, then the recovery procedure would yield sk_i.

Security There are several security requirements, that hold under a tolerable corruption (depends on the specifics of the scheme):

- *Owner's Key Safety.* The primary requirement is that, any secret key sk_i of a key-owner P_i must be totally hidden during the protocol executions.
- *Guardian's Secret Safety.* A guardian's secrets must be totally hidden during the protocol.
- *Worst Case Offline Attack.* In the worst-case, when all guardian's secrets are leaked to the attacker, still the attacker needs to run off-line dictionary attack to recover a secret.

- *Correct Recovery.* A recovery session either outputs the correct key, or $\perp$, but never a different key.

4 Bottom-Up Secret Sharing (BUSS)

Consider a set of parties $P_1, P_2, \ldots$, where each party P_i has an established identity denoted simply by integer i. We formalize a secret-sharing scheme for a $(t+1)$-out-of-n access structure, (first used in [4] to construct multiverse threshold signatures, albeit without any formalization), that allows each party to choose their shares independently of the other parties' shares and even the secret.

Syntax For $n, t \in \mathbb{N}$ such that $t+1 < n$, a bottom up secret sharing scheme for a $(t+1)$-out-of-n threshold access structure consists of algorithms with the following syntax:

- $\mathsf{Share}(s, \vec{\sigma}_B, B)$. The share algorithm takes a secret and $(n-1)$ shares, the corresponding set of indexes B ($i \notin B$ and $|B| = n-1$) to generate a public value φ.
- $\mathsf{Recon}(\varphi, \vec{\sigma}_R, R)$. The reconstruction algorithm takes $t+1$ shares $\vec{\sigma}_R$, the corresponding set of indexes R, and the public value φ to reconstruct a secret s (or outputs $\perp$ if the procedure fails).

Requirements The *correctness* requirement is straightforward, that is if for any secret s, any sets R, B such that $R \subseteq B$ and $|B| = (n-1), |R| = (t+1)$, any $\vec{\sigma}_B$ we have $\varphi \leftarrow \mathsf{Share}(s, \vec{\sigma}_B, B)$, then the following holds: $s \leftarrow \mathsf{Recon}(\varphi, \vec{\sigma}_R, R))$.

For *security* we require that, even if the attacker controls upto t shares (and may choose them too), the public information is statistically independent of the input s, that is, it does not leak any information about the secret.

4.1 Constructing BUSS

We construct a BUSS scheme for a $(t+1)$-out-of-n threshold access structure over a finite field $\mathbb{F}$ (e.g. $\{0,1\}^{256}$) as follows:

- $\mathsf{Share}(s, \vec{\sigma}_B, B)$:
  - Define a polynomial f over $\mathbb{F}$ of degree $(n-1)$ such that $f(0) := s$ and for all $j \in B$: $f(j) := \sigma_j$. Then output:

$$\varphi := \big(f(-1), \ldots, f\big(-(n-t-1)\big)\big)$$

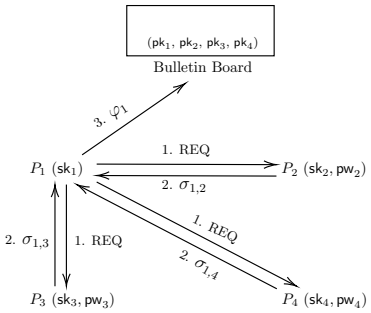
- $\mathsf{Recon}(\varphi, \vec{\sigma}_R, R)$:
  - Parse φ as $\big(f(-1), \ldots, f\big(-(n-t-1)\big)\big)$. Combine these $(n-$

$t-1$) points with $(t+1)$ points $\{f(j) := \sigma_j\}_{j \in R}$ ($|R| = t+1$) using Lagrange interpolation to output $s := f(0)$.

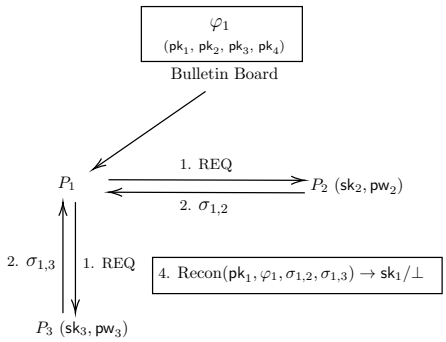
Satisfying Requirements The correctness follows from the Lagrange interpolation. The security can be argued by observing that at no point the attacker gets more than $(n-1)$ points of the polynomial, as long as it controls at most t shares, and observes the public output (which contains $(n-t-1)$ additional points). Hence, the polynomial would be totally hidden unconditionally.

5 KeyClub: Social Recovery in a Community of Key holder

Setting In this community setting we assume that each party P_i holds a key sk_i generated by executing a key-generation algorithm $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}_i(1^\lambda)$, where λ is a security parameter. The public keys are published on the bulletin board / blockchain in the beginning and also fix a threshold t (as a parameter) – every party P_i uses any set B of size $\geq (t+1)$ as guardians to back up key. Recovery is possible by collaborating with any subset $R \subseteq B$ of size $(t+1)$. Security holds even if (up to) any t of them are arbitrarily corrupt and even are colluding.



(a) Party P_1's key backup with P_2, P_3, and P_4



(b) Party P_1's key recovery with P_2 and P_3

Figure 1: Workflow of KeyClub ($n = 4, t = 1$) where $B = \{2,3,4\}$ and $R = \{2,3\}$. We denote the shares as $\{\sigma_{1,j}\}_{j \in \{2,3,4\}}$ and the public point as φ_1. Algorithm Recon reconstructs the secret sk_1 using the public point φ_1 and shares $(\sigma_{1,2}, \sigma_{1,3})$ and matches it with pk_1.

Key Generation All KeyGen algorithms that generate a key-pair $(\mathsf{sk}, \mathsf{pk})$ such that $\mathsf{sk} \in \mathbb{F}$ and $\mathsf{pk} \in \mathbb{G}$ for a cyclic group $\mathbb{G}$ with order $|\mathbb{F}|$. KeyGen has the following requirements.

1. We assume that for every pk there exists a unique sk such that $(\mathsf{pk}, \mathsf{sk})$ is a valid output of KeyGen.
2. One can efficiently verify whether a given $(\mathsf{pk}, \mathsf{sk})$ is a valid output of KeyGen.
3. It is computationally hard to guess sk given pk.

For example, one may consider keys of the form $(\mathsf{pk} = g^{\mathsf{sk}})$ for a cyclic group generated by g where discrete log is hard – this is used widely in many schemes, including BLS signatures, Schnorr signatures, El-Gamal encryptions etc. It is easy to check that this type of key generation *does* satisfy all of the above requirements.

We describe our main construction Π_{KC} or the *KeyClub* to socially recover keys in a community of key-holders for a threshold access structure. We use a hash function $\mathrm{H} : \{0,1\}^* \to \mathbb{F}$. The construction is provided below:

The Protocol KeyClub (Π_{KC})

– Π_{Back}: A party P_i runs this protocol as a key-owner with a set of $\geq (t+1)$ guardians $\{P_j\}_{j \in B}$ as follows:

- On request from P_i each guardian P_j computes $\sigma_{i,j} := \mathrm{H}(\mathsf{pk}_i, \mathsf{sk}_j, \mathsf{pw}_j)$ and sends that to P_i.
- The key-owner, on receiving $(\sigma_{i,j})_{j \in B}$, define $s := \mathsf{sk}_i$ and compute $\varphi \leftarrow \mathsf{Share}(s, (\sigma_{i,j})_{j \in B}, B)$ and publish $\mathsf{pub} := \varphi$.

– Π_{Rec}: A party P_i runs this protocol as a key-owner with a set of $(t+1)$ guardians $\{P_j\}_{j \in R}$ as follows:

- On request from the key-owner P_i each guardian P_j re-computes $\sigma_{i,j} := \mathrm{H}(\mathsf{pk}_i, \mathsf{sk}_j, \mathsf{pw}_j)$ and sends that to P_i.
- The key-owner, on receiving $(\sigma_{i,j})_{j \in R}$, retrieves $\varphi := \mathsf{pub}$ and computes $s \leftarrow \mathsf{Recon}(\varphi, (\sigma_{i,j})_{j \in R}, R)$.
- Then P_i checks whether (s_i, pk_i) is a valid key-pair. If yes then privately output $\mathsf{sk}_i := s_i$, else output $\perp$.

How the requirements are satisfied? The structural and correctness requirements are easy to see. The security requirements are satisfied as long as at most t parties are arbitrarily corrupt (and possibly colluding) because:

- *Owner's key safety* follows from the security of secret sharing. To formally prove this, one needs to assume that the hash function used behaves like a programmable random oracle plus rely on the fact that given the public key, it is computationally hard to obtain the corresponding secret key.

- *Guardian's secret safety* follows as long as the hash function is assumed to be a random oracle, because the output of hash then reveals no information on the input, as long as the input is unpredictable. The input contains guardian's secret key which is unpredictable given the public key. This also holds as long as the owner and $t-1$ other guardians are corrupt.
- *Worst-case offline attack* follows because, even if all guardian secrets are leaked, since passwords are never stored anywhere, the attacker is forced to guess the passwords correctly to compute the shares using dictionary attacks.
- *Correct Recovery* is ensued because in the end the key-owner checks the correctness of the recovered key with respect to the public key at the end of the recovery protocol.

6 RandRec: Socially Recovering Keys using VRF services

In this section we put forward another SKR scheme which works among parties $P_1, P_2 \ldots$ and a VRF server P_{vrf} who holds a secret key k_{vrf} (plus possibly a public verification key vk_{vrf} for verification). Here, we assume a key owner, P_i who generates a key-pair $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}_i(1^\lambda)$ which satisfies the Key generation criteria mentioned in the previous section. P_i publishes pk_i. However, here we assume a set of guardians who don't have any secret key themselves, such as Web2 users who are, for example, not native to the crypto ecosystem. We assume that the guardians just remember a password, but never store it. However, since the passwords are relatively easier to guess (that is they come from low entropy distribution), our BUSS scheme can not be used directly, because, by definition, our BUSS scheme requires the secret to be random (or to come from a high entropy distribution). Therefore, we assume a VRF server which is used by each guardian to derive a corresponding high (computationally) entropy shares. The rest of the protocol is pretty much the same as Π_{KC}. We call this protocol Π_{RR} or *RandRec*, which is described below. In the description, along with $\mathrm{H} : \{0,1\}^* \to \mathbb{F}$, we consider another hash function $\mathrm{H}' : \{0,1\}^* \to \mathbb{G}$.

Protocol RandRec (Π_{RR})

– Π_{Back} : Party P_i runs this protocol as a key-owner with a set of $\geq (t+1)$ guardians $\{P_j\}_{j \in B}$ as follows:

- On request from P_i each guardian P_j:
  - Sample a random $\rho \xleftarrow{\$} \mathbb{F}$
  - Computes $\mu_j := \mathrm{H}'(\mathsf{pw}_j)^\rho$ and send to P_{vrf} using secure channel.

- P_{vrf} responds with $z_j := \mu_j^{k_{\mathsf{vrf}}}$.
- P_j, on receiving z_j verifies the correctness with respect to vk_{vrf}. It aborts if the check fails. Otherwise, it computes $\mathsf{sk}_j := z_j^{1/\rho}$.
- P_j then computes $\sigma_{i,j} := \mathrm{H}(\mathsf{pk}_i, \mathsf{sk}_j, \mathsf{pw}_j)$ and sends that to P_i.

• The key-owner, P_i, on receiving $(\sigma_{i,j})_{j\in B}$, define $s := \mathsf{sk}_i$ and compute $\varphi \leftarrow \mathsf{Share}(s, (\sigma_{i,j})_{j\in B}, B)$ and publish $\mathsf{pub} := \varphi$.

– Π_{Rec}: A party P_i runs this protocol as a key-owner with a set of $(t+1)$ guardians $\{P_j\}_{j\in R}$ as follows:

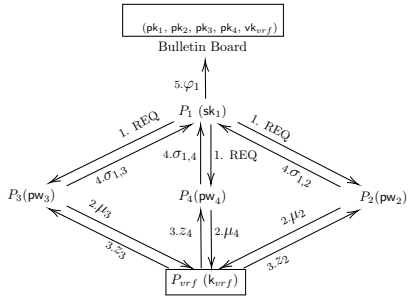
• On request from P_i each guardian P_j re-computes $\sigma_{i,j}$ as:
- Sample a random $\rho \xleftarrow{\$} \mathbb{F}$
- Computes $\mu_j := \mathrm{H}'(\mathsf{pw}_j)^\rho$ and send to P_{vrf} using secure channel.
- P_{vrf} responds with $z_j := \mu_j^{k_{\mathsf{vrf}}}$.
- P_j, on receiving z_j verifies the correctness with respect to vk_{vrf}. It aborts if the check fails. Otherwise, it computes $\mathsf{sk}_j := z_j^{1/\rho}$.
- P_j then computes $\sigma_{i,j} := \mathrm{H}(\mathsf{pk}_i, \mathsf{sk}_j, \mathsf{pw}_j)$ and sends that to P_i.

• The key-owner, on receiving $(\sigma_{i,j})_{j\in R}$, retrieves $\varphi := \mathsf{pub}$ and computes $s \leftarrow \mathsf{Recon}(\varphi, (\sigma_{i,j})_{j\in R}, R)$.

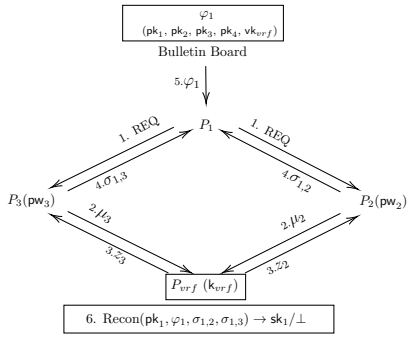
• Then P_i checks whether (s_i, pk_i) is a valid key-pair. If yes then privately output $\mathsf{sk}_i := s_i$, else output $\perp$.

How the requirements are satisfied? The structural and correctness requirements are easy to see. For security requirements observe that:

- *Owner's key safety* can be seen exactly the same way as Π_{KC}.
- *Guardian's Secret safety* follows from two things: (i) due to the blinding factor ρ, the VRF computation is input-oblivious [6] and hence even if the VRF server is corrupt, it does not know anything on a guardian's secret, such as passwords; (ii) similar to Π_{KC}, if the hash function H is assumed to be a random oracle, the output of hash then reveals no information on the input, as long as the input is unpredictable, which is guaranteed by the unpredictability of the VRF output. This also holds as long as the owner and $t-1$ other guardians are corrupt.
- *Worst-case offline attack* follows, since passwords are never stored anywhere; so the attacker must guess the passwords correctly to compute the shares using dictionary attacks even if all guardian's secrets are leaked. This also holds when the VRF server is corrupt.
- *Correct Recovery* is ensured as the key-owner checks the correctness of the recovered key with respect to the public key at the end of the recovery pro-



(a) Party P_1's key backup with P_2, P_3, and P_4



(b) Party P_1's key recovery with P_2 and P_3

Figure 2: Workflow of RandRec ($n = 4, t = 1$) where $B = \{2, 3, 4\}$ and $R = \{2, 3\}$. Here, $\{\mu_j\}_{j\in\{2,3,4\}}$ and $\{z_j\}_{j\in\{2,3,4\}}$ denote the input and output of the VRF. We denote the guardian passwords as $\{\mathsf{pw}_j\}_{j\in\{2,3,4\}}$, the shares as $\{\sigma_{1,j}\}_{j\in\{2,3,4\}}$, and the public point as φ_1. Algorithm Recon reconstructs the secret sk_1 using the public point φ_1 and shares $(\sigma_{1,2}, \sigma_{1,3})$ and matches it with pk_1.

tocol. Furthermore, the verifiability of the VRF output would ensure that its response is verified immediately after receiving. This gives additional robustness against a corrupt VRF server.

Distributed VRF Our description considers a single VRF server. However, it is readily compatible with a distributed VRF (dVRF) service with many servers as well. In fact, a dVRF service would be more robust, because VRF computation is essential both during backup and recovery procedures.

7 Rotating Shares for the Same Key

In our basic schemes, we assume that the shares $\sigma_{i,j}$ are deterministic for a specific key. This is because, once the owner's key sk_i is fixed, the share $\sigma_{i,j}$

for guardian with identity-j is derived deterministically by hashing a number of components all of which are fixed. Therefore, backing up each secret multiple times would yield the same share for the same guardian, irrespective of the backup set. Realistically, it may be desirable to have support for share rotation – which means even if the same secret key sk_i is shared multiple times, different shares σ_{i,j,t_1} and σ_{i,j,t_2} should be generated for guardian-j. A simple way to enable that would be to include a counter t, which is maintained by the key owner. Then use t additionally to derive $\sigma_{i,j,t} := \mathrm{H}(\mathsf{pk}_i, \mathsf{sk}_j, \mathsf{pw}_j, t)$. However, this would burden the key owner who now needs to store the counter. This can be dispensed with in a synchronized setting, in that everyone has access to a clock, and t is just the current time stamp.

References

[1] Cypherock – The world's first hardware wallet without a seed phrase backup. `https://docs.cypherock.com/`.

[2] Dash: A Privacy Centric Crypto Currency. `https://www.exodus.com/assets/docs/dash-whitepaper.pdf`.

[3] MetaMask Developer Documentation. `https://docs.metamask.io/`.

[4] Leemon Baird, Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Threshold signatures in the multiverse. Cryptology ePrint Archive, Paper 2023/063, 2023. `https://eprint.iacr.org/2023/063`.

[5] Vitalik Buterin. Why we need wide adoption of social recovery wallets, 2021. https://vitalik.eth.limo/general/2021/01/11/recovery.html.

[6] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. SoK: Oblivious pseudorandom functions. Cryptology ePrint Archive, Report 2022/302, 2022.

[7] Panagiotis Chatzigiannis, Konstantinos Chalkias, Aniket Kate, Easwar Vivek Mangipudi, Mohsen Minaei, and Mainack Mondal. Sok: Web3 recovery mechanisms. *IACR Cryptol. ePrint Arch.*, page 1575, 2023.

[8] Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. pages 651–668, 2019.

[9] Yehuda Lindell. Cryptography and mpc in coinbase wallet as a service (waas), 2023.

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

[11] New York Times. Tens of billions worth of bitcoin have been locked by people who forgot their key. `https://www.nytimes.com/2021/01/13/business/`

tens-of-billions-worth-of-bitcoin-have-been-locked-by-people-who-forgot-their-key.html.