ANARKey: A New Approach to (Socially) Recover Keys

Aniket Kate
Purdue University / Supra Research
USA
aniket@purdue.edu

Pratyay Mukherjee
Supra Research, India
pratyay85@gmail.com

Hamza Saleem
Supra Research, USA
h.saleem@supraoracles.com

Pratik Sarkar
Supra Research, India
iampratiksarkar@gmail.com

Bhaskar Roberts
University of California, Berkeley
USA
bhaskarr@berkeley.edu

ABSTRACT

In a social key recovery scheme, users back up their secret keys (typically using Shamir's secret sharing) with their social connections, known as a set of guardians. This places a heavy burden on the guardians, as they must manage their shares both securely and reliably. Finding and managing such a set of guardians may not be easy, especially when the consequences of losing a key are significant.

We take an alternative approach of social recovery within a community, where each member already holds a secret key (with possibly an associated public key) and uses other community members as their guardians forming a mutual dependency among themselves. Potentially, each member acts as a guardian for upto $(n-1)$ other community members. Therefore, in this setting, using standard Shamir's sharing leads to a linear ($O(n)$) blow-up in the internal secret storage of the guardian for each key recovery. Our solution avoids this linear blowup in internal secret storage by relying on a novel secret-sharing scheme, leveraging the fact that each member already manages a secret key. In fact, our scheme does not require guardians to store anything beyond their own secret keys.

We propose the first formal definition of a social key recovery scheme for general access structures in the community setting. We prove that our scheme is secure against any malicious and adaptive adversary that may corrupt up to t parties. As a main technical tool, we use a new notion of secret sharing, that enables $(t+1)$ out of n sharing of a secret even when the shares are generated independently – we formalize this as bottom-up secret sharing (BUSS), which may be of independent interest.

Finally, we provide an implementation benchmarking varying the number of guardians both in a regional, and geo-distributed setting. For instance, for 8 guardians, our backup protocol takes around 146-149 ms in a geo-distributed WAN setting, and 4.9-5.9 ms in the LAN setting; for recovery protocol, the timings are approximately the same for the WAN setting (as network latency dominates), and 1.2-1.4 ms for the LAN setting.

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 on cryptography, creating secure and reliable (that is available when needed) wallet services, for storing keys, has garnered substantial attention [10, 17] 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 a transaction requires a signature using the corresponding secret-key, then those funds are lost forever if the secret key cannot be recovered. It is estimated that USD 140 billions worth of BTC is unrecoverable due to lost secret keys [26]! Therefore, many existing wallets support a backup option, either via mnemonic pass phrases [14], which one may write down in a secure place, or splitting the key [11, 19] using simple secret sharing (such as Shamir's [23]) 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 [12] or hardware wallets [29]) incidental memory erasure or losing passwords (or a combination) [1] may happen realistically invoking a loss of funds. In fact, in scenarios involving permanent disappearance, such as the death or disability of the key-owner, a similar loss of funds can take place.

Many of these issues are mitigated in social recovery solutions, which, as laid out by Buterin [8], 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*, to back 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. Importantly, in the event of a permanent disappearance of the key-owner, e.g. demise, the legitimate nominee can coordinate with any $(t+1)$ guardians to recover the secret key and thus inherit any asset locked with the key.

However, the security and reliability aspects of the social recovery approach crucially relies on the guardians. An ideal guardian should be technologically adept, trustworthy, and reliable. Finding

Bhaskar Roberts worked on this project primarily during a summer internship at Supra Research.

[1]In [8], 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).

a set of guardians all of whom posses these attributes to a reasonable degree may not be easy. Importantly, since the guardians do not have any stake (barring just helping the key owner) with the existing approaches, it may be too optimistic to assume that they would store the shares both securely and reliably.

Motivated by this, we put forward a new *community-based model for social recovery.* Our model considers a community of secret-key holders (e.g., users of cryptocurrency), such that everyone can use (a subset of) the other community members as their guardians. In return, they are also expected to serve as guardians for other parties. This creates an ecosystem of mutual dependence, in that there is a clear motivation for each member to store the shares of other key owners securely and reliably, with the hope that the other members would also return the favor.

Using a simple $(t+1)$-out-n secret sharing in this model, 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. Now, in the community setting, everyone would potentially use other members as guardians, and hence it is expected that party P_j may have to serve as a guardian for m different key-owners. In that scenario, P_j has to manage m different shares securely and reliably. The requirement of securely and reliably many shares (scales with $O(m)$) becomes challenging for the members, and even worse, this may end up discouraging users from joining the community.

To resolve this issue we propose a new solution, in that, parties *do not need to store any additional data* beyond their own secret key securely and reliably. Our main idea is to leverage the fact that each user already maintains a secret-key, and the shares would be derived from that without hurting security. In fact, our protocol is quite general and works beyond the community setting: as long as each guardian can manage a single secret key, there is *no need to store any additional information*, regardless of the number of key owner the guardian supports.

1.1 Our Contribution

In short, our contributions are:

- We introduce the concept of community-based social recovery. We formalize the correctness and security definitions of community-based social key recovery schemes in a stand-alone simulation-based framework [9, 16]. Intuitively, our definition ensures that no computationally bounded adversary, which corrupts parties (maliciously and adaptively) satisfying some access structure is able to distinguish between a real world where the actual protocol is run and an ideal world, where the honest party's responses are computed without their secret inputs. To the best of our knowledge, this is the *first* formalization of *any* social key recovery scheme.
- We design a simple protocol for community-based social key recovery. Each owner's key can be backed up with a subset of the rest of the community members, denoted as guardians. The guardians do not have to store anything in addition to their own key. For recovery, any $(t+1)$ guardians must help correctly for a pre-defined threshold t. The scheme supports every party to back up their respective keys with (a subset of) everyone else, without needing to store anything apart from their own secret key – this establishes an ecosystem of mutual dependence without additional overhead. Our protocol is secure (with abort) against any *malicious* and *adaptive* corruption up to t parties. 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.[2]
- The main technical tool, we rely upon, is a new type of secret sharing scheme, which supports independent shares for a $(t+1)$-out-of-n threshold access structure – given a secret, and $(n-1)$-many independently chosen shares, one can produce a (set of) public values, such that during reconstruction, any $(t+1)$ of the shares and the public values can be combined to reconstruct the secret. This type of secret sharing has been used recently by Baird et al. [3] to design multiverse threshold signatures. In this paper we formalize this as bottom-up secret sharing (BUSS) with an adaptive simulation security definition – this may be of independent interest.
- To demonstrate practicality we perform extensive benchmarking, varying the tools (such as the types of elliptic curves and hash functions) and the number of guardians both in a regional, and geo-distributed setting. For instance, with standard hash functions, for 8 guardians, our backup protocol takes around 146-149 ms in a geo-distributed WAN setting (with a network delay of 138 ms), and 4.9-5.9 ms in the intra-regional LAN setting; for recovery protocol, the timings are approximately the same for WAN setting (as communication latency dominates), and 1.2-1.4 ms for the LAN setting. We also provide more optimized numbers with multi-threading that enhances the performances significantly (see Section 7 for details).

Additionally, we discuss a few extensions of our key recovery scheme to accommodate settings where a guardian may update their secret key, or maintain a separate "guardian-key" in addition to its signing key. We further demonstrate that our scheme can be made to work on top of most cold wallets, such as Ledger [15], Trezor [29], where the guardians store their secret keys in the cold wallet and use that only for producing deterministic signatures.

1.2 Technical Overview

Let us first describe our community setting, in that there are N parties $P_1, \ldots, P_N$ pairwise connected via secure and authenticated channels. Also we assume a reliable public storage, e.g. a bulletin board. We assume that each party P_i holds a key sk_i generated by executing a key-generation algorithm $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$,

[2]As elaborated by Buterin [8], guardians not knowing each other intuitively reduces the possibility of collusion. In our community setting consisting of N members, a key owner can use any subset (say, of size n) of the members as her guardians. So there are $\binom{N}{n}$ many such possibilities, which means guessing the set of guardians has a low probability of success for adequately chosen n, for instance, $n = N/2$

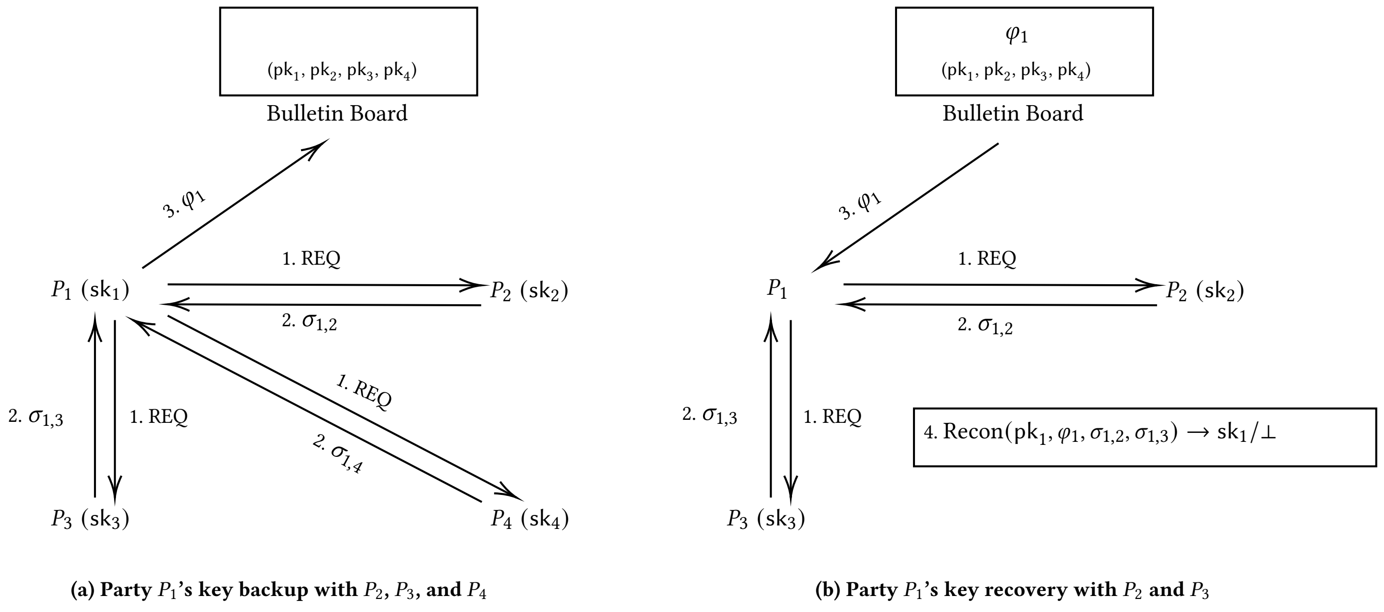
$(\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4)$
Bulletin Board
3. φ_1
1. REQ
P_1 (sk_1)
P_2 (sk_2)
2. $\sigma_{1,2}$
1. REQ
2. $\sigma_{1,4}$
2. $\sigma_{1,3}$
1. REQ
P_3 (sk_3)
P_4 (sk_4)
(a) Party P_1's key backup with P_2, P_3, and P_4
φ_1
$(\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4)$
Bulletin Board
3. φ_1
P_1
1. REQ
P_2 (sk_2)
2. $\sigma_{1,2}$
2. $\sigma_{1,3}$
1. REQ
4. $\mathsf{Recon}(\mathsf{pk}_1, \varphi_1, \sigma_{1,2}, \sigma_{1,3}) \to \mathsf{sk}_1/\bot$
P_3 (sk_3)
(b) Party P_1's key recovery with P_2 and P_3

Figure 1: Workflow of our social key recovery scheme $(n = 4, t = 1)$ 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.

where λ is a security parameter.[3] The public keys are published on the bulletin board in the beginning and serve as identities. For simplicity we use integers $i \in [N]$ to represent party P_i's identity. Also, fix a threshold t (as a parameter) and assume that every party uses exactly $n-1 > t$ $(n < N)$ other parties as guardians – our protocol also works if each key-owner chooses a different n satisfying $n-1 > t$.

Now, when P_i wishes to backup her secret key sk_i (for which a public key pk_i is publicly known), she selects a set $B \subseteq [N] \setminus \{i\}$ of $n-1$ guardians $\{P_j\}_{j\in B}$. As mentioned earlier, a simple $(t+1)$-out-of-n secret sharing using a t-degree polynomial would yield to a share $\sigma_{i,j}$ (for guardian P_j), which is uncorrelated with P_j's own secret key sk_j. Our goal is to use sk_j to derive $\sigma_{i,j}$ for any i instead. At first glance, this looks hard, because deriving each $\sigma_{i,j}$ from independently sampled sk_j's would lead to independent shares $\sigma_{i,j}$'s – in general this leads to a n-out-of-n (or additive) secret sharing. However, here we use a technique recently used by Baird et al. [3]. The idea is to first derive $\{\sigma_{i,j}\}_{j\in[n-1]}$ independently (deterministically using corresponding sk_j), which together with the key-owner's secret sk_i defines an $(n-1)$-degree polynomial. Nevertheless, making another $(n-t-1)$ points on the polynomial public reduces the "effective threshold" of the system to $(t+1)$. These additional points would be derived via interpolation by the key-owner P_i herself, once she gets back all $\{\sigma_{i,j}\}_{j\in B}$, and then those points will be made public. [4] A couple of issues still remain: (i) a guardian P_j must ensure that $\sigma_{i,j}$ does not leak any information about sk_j; (ii) the $(n-t-1)$ public points must be computed on inputs that are different from any input corresponding to $\sigma_{i,j}$. The first issue is fixed by using the following derivation $\sigma_{i,j} := \mathrm{H}(i, \mathsf{sk}_j)$ (alternatively $\sigma_{i,j} := \mathrm{H}(\mathsf{pk}_i, \mathsf{sk}_j)$) – assuming H is a random oracle this gives no information about sk_j, as long as sk_j is hard to compute given pk_i (for example, the public key in BLS signature [7] and ECDSA signature this is true due to the hardness of computing discrete log). The second issue is resolved by using negative evaluation points $-1, -2, \ldots, -(n-t-1)$, akin to Baird et al. [3]. The overall backup protocol can be summarized as follows:

- The key-owner P_i chooses a set $B \subseteq [N] \setminus \{i\}$ and reaches out to the guardians P_j for all $j \in B$.
- Each guardian P_j, on receiving the request, computes $\sigma_{i,j} := \mathrm{H}(i, \mathsf{sk}_j)$ and sends that back to P_i.
- P_i, on receiving $\{\sigma_{i,j}\}_{j\in B}$, computes a $(n-1)$-degree polynomial f_i (over a field, where the secrets sk_i lie) by setting: $f_i(0) = \mathsf{sk}_i$ and for all $j \in B$: $f_i(j) = \sigma_{i,j}$. Then it evaluates $f_i(-1), f_i(-2), \ldots, f_i(-(n-t-1))$ and publishes them.

Given the above protocol, the recovery protocol works simply by requiring the key-owner P_i to interact with $(t+1)$ guardians $\{P_j\}_{j\in R}$ for $R \subseteq B$ and $|R| = t+1$ as follows:

- P_i reaches out to any $(t+1)$ guardians $\{P_j\}_{j\in R}$.
- Each guardian recomputes $\sigma_{i,j} := \mathrm{H}(i, \mathsf{sk}_j)$ and sends back.
- On receiving $(t+1)$ shares $\{\sigma_{i,j}\}_{j\in R}$, P_i interpolates $\mathsf{sk}_i := f_i(0)$ using n distinct evaluation points combining these $(t+1)$ evaluation points $f_i(j) := \sigma_{i,j}$ and $(n-t-1)$ public evaluation points $f_i(-1), f_i(-2), \ldots, f_i(-(n-t-1))$. In the

[3]For example, this can be a key generation algorithm of BLS signature, where $\mathsf{pk} = g^{\mathsf{sk}}$ for a group generator g. Also we note that, our scheme works even if parties use different KeyGen, as long as the key pair satisfies certain basic conditions, as described in Section 6. For simplicity, in this paper, we consider that every party executes the same KeyGen.

[4]We observe that for each backup session a fresh set of $(n-t-1)$ public points are generated. Storing all of these on the blockchain would require spending substantial gas cost, which scales linearly with the number of backups. Instead, one may just put a hash of the points, and store the point themselves off-chain.

end it checks whether sk_i is the correct key with respect to public pk_i.

Figure 1 provides a workflow of our scheme for the case where $n = 4$ and $t = 1$. Party P_1 first creates a backup of key sk_1 by interacting with parties P_2, P_3, and P_4, as illustrated in Figure 1a. The public evaluation shares φ_1 are published on the bulletin board. In the event of key loss, party P_1 can interact with $t + 1$ parties i.e. P_2 and P_3 in this example, and the public bulletin board to recover the key, as shown in Figure 1b.

Let us remark a few important things about our protocols: first, note that as long as adversary corrupts at most t parties then it gets t shares. Given $(n-t-1)$ public points, the adversary knows at most $(n - 1)$ evaluation points, insufficient to recover any information about sk_i. Second, if P_j works as a guardian for different key owners $i_1, \dots, i_m$, all shares $\sigma_{i_1,j} = \mathrm{H}(i_1, \mathsf{sk}_j), \dots, \sigma_{i_m,j} = \mathrm{H}(i_m, \mathsf{sk}_j)$ are uncorrelated and independent as long as sk_j is hidden, assuming H is a random oracle. Thirdly, $\sigma_{i,j}$ does not leak anything about sk_j as long as sk_j is hard to predict given pk_j, due to random oracle properties of H as well. Furthermore, here the guardians do not need to know each other, as B is never made public – as mentioned in [8], this reduces the chance of collusion, as first they need to figure out the set B. We note that, in our protocol, during recovery a key-owner must remember the backup set B, as she needs to reach out to a $(t + 1)$ size subset of that set. This is relatively easy information to remember, as also pointed out in [8], and plausibly is necessary too.

Malicious security comes virtually for free. This is because, a malicious key-owner can only hurt herself, and thus can not do worse than a semi-honest key-owner. A malicious guardian may send arbitrary computation at either the backup or recovery phase. But that would still leave enough entropy to the secret. An inconsistent behavior, such as sending different values at the time of backup and recovery would yield a faulty recovery of sk_i. But since this is checked against the public key pk_i at the very last step of the recovery protocol, the key-owner would be able to catch this behavior and abort. However, the key-owner would not be able to identify the cheater as our protocol does not support identifiable abort – we leave that as an interesting open question for the future.

Our protocol can be proven secure against adaptive corruption, where the adversary may corrupt parties at any time during the execution. The main challenge to achieve this, with respect to a simulation-based definition, such as ours, is the following scenario: the simulator simulates an honest party P_j's response, say $\sigma_{i,j}$ in a backup protocol initiated by a corrupt key-owner P_i, without its secret key sk_j, and then later P_j gets corrupted leaking the entire secret state of P_j – at that point, the simulator (who now also obtains sk_j) needs to ensure that the leaked secret state is consistent with the prior responses. This is handled by programming the random oracle to $\sigma_{i,j} := \mathrm{H}(i, \mathsf{sk}_j)$ adaptively. In another scenario, when a key-owner P_i gets adaptively corrupt, after a backup session, then too such programming is needed. However, now the simulator needs to ensure that the public points, published during the backup session, are consistent with the secret key sk_i. In this case, the simulator just samples uniform random (and hence independent) evaluations for public points. Since not more than t corruptions are allowed, even an unbounded adversary does not obtain more than $(t+1)$ points (it can obtain sk_i from pk_i by, for example, brute-force). Once sk_i is available, the honest party's responses to the prior backup session, available in the secret state of P_i, needs to be computed via interpolation (this is in contrast to the actual protocol, where honest shares are computed independently, and public points are interpolated). But, due to the properties of secret sharing, the two procedure are identical, as long as up to t parties are (adaptively) corrupt. We formalize this secret sharing scheme as *bottom up secret sharing* (see Section 5), and formalize this security requirement as *perfect adaptive simulation security*. This formalization maybe of independent interest.

1.3 A Domino Effect for variable thresholds

In this paper, we only consider a fixed t for our design, though our definition allows arbitrary access structures. We notice an interesting feature for variable threshold access structure in our protocol, which is akin to the so-called domino effect. Let us consider an access structure where each party P_i has threshold t_i, and $t_{i+1} = (t_i + 1)$. Now, if the adversary corrupts $(t_1 + 1)$ parties, then it may obtain the secret key of P_1 since in our protocol, given a secret key, all corresponding shares can be computed deterministically. So, now it has $(t_1 + 2) = (t_2 + 1)$ secret keys. Again, it may well be possible now that those $(t_2 + 1)$ keys are sufficient to recover the secret key of P_2. Continuing like this one may end up recovering all secrets just by corrupting $t_1 + 1$ parties. However, in our definition, this does not show up because we do not allow corruption of more than t_1 parties. Also note that, if parties have more unknown variables, such as passwords, or other hidden randomnesses, such effects can be mitigated, at least partially. We leave further exploration of variable threshold access structure as an interesting future work.

1.4 Roadmap

We present the related works in the literature in Section 2. We present the necessary preliminaries in Section 3. We formally define Social Key recovery in Section 4. Then we define and construct BUSS in Section 5. Using BUSS we construct our Social Key Recovery protocol in Section. 6. We also implement our scheme. We benchmark in both regional and geo-distributed settings and present evaluations in Section 7. Finally, we briefly describe a number of extensions in Section 8.

2 RELATED WORK

Smart-contract based Social Recovery [8, 20]. In [8] Buterin refers to a social key recovery protocol, that is operated via a smart contract, which is executed when a recovery is initiated by the key-owner, the guardians sign a special transaction that recovers the fund. However, this is a non-cryptographic solution, and the original secret key is not recovered. In contrast, our solution enables the key-owner to recover the original secret-key, which may be used for any desired purpose, subsequently. So, from that perspective, our solution is more general and uses cryptography instead of relying on the smart-contract's capability. The only functionality we need from the blockchain is to support an immutable bulletin board that would store the public points, or their hashes, reliably. A similar solution was proposed in [20], which additionally uses encryption of shares where the decryption key is further split into shares. Nevertheless, this also enables recovering funds instead of secret keys. Also, both of these solutions suffer from scalability issues in our community setting where the same guardian is used by many key-owners.

There are other key-recovery / account recovery designs that consider mainly a client-server setting, in that servers are typically assumed to store secret information per user, can do much heavier computation than the clients, and also are available during recovery. Those solutions are not directly compatible in our social recovery setting in a community but may offer new technical insights for future adaptation. We discuss a few of them below.

Coinbase WaaS: Key-recovery using MPC [17]. Recently in a whitepaper [17], Lindell describes a key recovery solution for Coinbase's wallets-as-a-service or WaaS. The whitepaper describes two designs for key-recovery using interactive protocols without formalization. In the first design, the secret key is encrypted[5], then uploaded in the cloud and the decryption key is stored in a secure enclave. Together, the ciphertext and decryption key are sufficient to recover the wallet key. In their second design, the decryption key is split into two additive shares, one held by the wallet-owner (i.e. key-owner) in the cloud and one held by Coinbase server – as stated in the whitepaper, this design is easier to use and has more resilience against loss because the user does not need to store any information on their local device. Nevertheless, the designs are not really compatible with the social recovery paradigm, because they fundamentally rely on existence (and availability) of a reliable server maintained by Coinbase, that receives and stores back-up information from each user and participates in signing every transaction from the user. Compared to that, our design relies on a community with an access structure, that can be any $(t+1)$ of the whole set of guardians. Furthermore, the guardians are not relied upon to store backup information, such as ciphertexts or random shares, from the user – they only store their own secret keys. Moreover, they only need to interact with the user during back-up and recovery – so the internal secret storage of a guardian does not increase with the number of supported backups.

Account Recovery for a Privacy-Preserving Web Service [18]. Account recovery with a server has been considered in a privacy-preserving setting, where the account information, such as user names, passwords, or even the security questions must be hidden from the server. This solution uses cryptographic techniques such as oblivious PRFs along with rate limiters to ensure privacy but relies on the user remembering the information such as passwords, security questions, etc, and relies on external servers.

There are many more key / account recovery approaches, e.g. off-chain backup using 2FA [2], pre-signed transactions [25] which are applicable in settings that are even more different from ours. For a comprehensive study we refer to the SoK [10]. There are protocols [2] that rely on cloud services and allows the user to backup-recover their keys via 2-factor authentication methods. In the BitGo recovery scheme [5] the user generates a backup key and main wallet key. The user needs to store a "keycard" that contains an encrypted version of the main and backup key. To recover funds the user produces the keycard combined with a user passphrase and other personal inputs to build a transaction that sends the account funds to a new address. There are other [6] recovery schemes that allow funds recovery. The schemes of Sequence [22] and Torus [28] also generate multiple keys during the account creation. These keys are used to recover the account upon loss of the main signing key. Zengo [30] creates secret shares of the wallet key and stores one share on the user's device and the other share is stored securely (encrypted under the Zengo public key) with the Zengo servers. To execute a transaction, both secret shares are used to sign. To recover the key, the user needs to authenticate itself via biometrics and then recover the key using the two shares.

Login using Web2 methods: Mysten Lab's ZKLogin [24], Aptos KeyLess [1]. In these two very similar designs, the high-level idea is to use Web2 authentication methods to enable Web3 authentication. The biggest benefit of these approaches is that the user can just be a native Web2 user, without ever requiring to manage any key. So, the need for any wallet is totally dispensed with. While we recognize that these approaches would indeed be greatly useful to bring more Web2 user onboard to Web3 ecosystems, they still rely on servers controlled by large-scale corporations such as Google and hence is not really compatible with the decentralization paradigm of Web3. So, any social recovery approach, including our approach may be viewed as a complementary solution for a community where users are already native to the Web3 ecosystem.

Adept Secret Sharing [4]. Bellare, Dai and Rogaway proposed the notion of adept secret sharing in [4]. In addition to the standard guarantee such as privacy, adept secret sharing offers new properties such as authenticity and error correction. Their authenticity notion guarantees that a party can verify whether a reconstructed secret is correct or not. This is achieved by storing a commitment of the secret in a reliable public storage ahead of time, and then check with that after reconstruction. Looking ahead, our social recovery scheme achieves malicious security in a similar way by checking with the public key which is essentially a commitment of the secret key, though we do not include this property in our notion of bottom-up secret sharing. Moreover, their construction uses a PRF to generate the coefficients of the secret polynomials, whereas our construction (Section 5.1) uses a hash function to define the evaluation points of the secret polynomial. Nevertheless, despite some technical similarities, they did not consider a bottom-up approach.

3 NOTATION AND PRELIMINARIES

Notations. We use $\mathbb{N}$ to denote a set of positive integers, and $[n]$ to denote the set $\{1, \dots, n\}$ for any $n \in \mathbb{N}$. We denote the security parameter by $\lambda \in \mathbb{N}$. A set $X = \{x_1, \dots, x_n\}$ 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 $(x_1, \dots, x_n)$ 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 λ as an input, even if not always mentioned explicitly; all definitions work for sufficiently large choice of λ. We use negl to denote a negligible function, which is defined to be a function $f : \mathbb{N} \to \mathbb{R}$, such that for every polynomial p, there exists an $n \in \mathbb{N}$ such that for all $\lambda > n$, it holds that $f(\lambda) < 1/p(\lambda)$. We

[5]In more detail, the wallet key is split into two shares, and both shares are encrypted.

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 undefined value, or invalidity.

We model computationally bounded adversaries by probabilistic polynomial time (PPT) algorithms. We also consider our adversaries to be interactive algorithms $\mathcal{A}$, which can be rigorously modeled as interactive Turing machines. An algorithm $\mathcal{A}$ with oracle access to an oracle O is denoted as $\mathcal{A}^{O(\cdot)}$. We say a particular problem is *computationally hard* to imply that for any PPT adversary, the probability of solving a random instance of that problem is bounded by $\mathsf{negl}(\lambda)$.

Adversarial Model. We consider adaptive and malicious PPT adversaries. By malicious we mean that the adversary can behave arbitrarily in the protocol. Looking ahead, security against malicious adversary basically comes for free, as a malicious adversary, at best, can invoke an abort. Adaptive adversaries can corrupt parties at any time during the execution – this model of corruption is much more realistic than a weaker static adversarial model, where the corrupted set of parties does not change since the start of the execution. Handling adaptive security in our protocol is more challenging. We also assume there is no erasure, so any party's secret state can only be appended, but not deleted – in this setting adaptive corruptions are particularly difficult to analyze, because in a simulation any honest party's secret state at any point must be "explained" consistently with respect to prior messages, if that party gets corrupted at a later point during the execution. Note that the "no erasure" assumption does not contradict the essence or utility of a key recovery protocol, because recovery takes place rather infrequently, typically for accidental deletion / loss of the secret key.

4 DEFINITION

In this section we present our formal definitions. Note that, the social recovery scheme defined below does not require the guardians to remember any secret back-up information other than their own a priori generated sk using KeyGen. It suffices for the back-up procedure Π_{Back} to produce only *public* back-up information pub. Also, we define the scheme for a setting when there is a single KeyGen. But it is straightforward to extend to a setting with multiple party-specific key-generation procedures.

Definition 4.1 (Social Key Recovery Scheme (SKR))**.** Consider N parties $P_1, \ldots, P_N$, and define an access structure $\mathfrak{A}$ which consists of pair of sets (B, R) such that $B \subseteq [N]$ and $R \subseteq B$. We assume without loss of generality that each party P_i has an established public identity-i.[6] Let KeyGen be a key generation algorithm, which produces a pair of keys $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. Then a social recovery scheme Π_{SKR} between N parties $P_1, \ldots, P_N$ for KeyGen and access structure $\mathfrak{A}$ consists of a triple of protocols $(\Pi_{\text{Init}}, \Pi_{\text{Back}}, \Pi_{\text{Rec}})$ executed among a subset of the parties in the following order: first Π_{Init} is executed by all parties; then each party may initiate an instance of Π_{Back} once with a subset of parties in $B \subseteq [N]$; finally once P_i has finished executing Π_{Back}, it can initiate Π_{Rec} arbitrarily many times. The protocols have the following syntax:

- Π_{Init}. In this protocol, a party P_i locally generates a key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ either computing $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$ or using another algorithm, specified by the protocol.[7] Each party P_i publishes pk_i. An execution is denoted by:
$$(\vec{\mathsf{pk}}_{[N]}, \vec{\mathsf{sk}}_{[N]}) \leftarrow \Pi_{\text{Init}}(1^\lambda, 1^N)$$
Once this is completed, the below protocols may be executed by each party P_i with the same public input $\vec{\mathsf{pk}}_{[N]}$.
- 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 uses secret key sk_j. The protocol concludes with a *public* back-up string pub_i. We denote such execution by:
$$\mathsf{pub}_i \leftarrow \Pi_{\text{Back}}(i, B, \vec{\mathsf{pk}}_B, \vec{\mathsf{sk}}_{B \cup \{i\}})$$
- Π_{Rec}: If P_i wishes to recover sk_i using a recovery set R, she runs this protocol without any secret input with a set of guardians $\{P_j\}_{j \in R}$, each of which uses their secret key sk_j. In addition, pub_i may be used by all parties. At the end of this protocol, the key-owner may receive a private output sk_i (or $\perp$ if unsuccessful). One such execution is denoted as:
$$\mathsf{sk}_i/\perp \leftarrow \Pi_{\text{Rec}}(i, R, \vec{\mathsf{pk}}_R, \mathsf{pub}_i, \vec{\mathsf{sk}}_R)$$

Correctness. For correctness we require that for any sufficiently large λ, any $i \in [N]$, any $(B, R) \in \mathfrak{A}$:
$$\Pr\left[\mathsf{sk}_i \leftarrow \Pi_{\text{Rec}}(i, R, \vec{\mathsf{pk}}_R, \mathsf{pub}_i, \vec{\mathsf{sk}}_R) \,\middle|\, \begin{array}{l}(\vec{\mathsf{pk}}_{[N]}, \vec{\mathsf{sk}}_{[N]}) \leftarrow \Pi_{\text{Init}}(1^\lambda, 1^N);\\ \mathsf{pub}_i \leftarrow \Pi_{\text{Back}}(i, B, \vec{\mathsf{pk}}_B, \vec{\mathsf{sk}}_{B\cup\{i\}})\end{array}\right] = 1$$

Security. For the security of Π_{SKR} for a given KeyGen and $\mathfrak{A}$, we need that for any sufficiently large $\lambda \in \mathbb{N}$ there exists a PPT simulator $\mathcal{S}$ in the ideal world, such that for any N and any PPT adversary $\mathcal{A}$ we have that:
$$\mathsf{Real}_{\mathcal{A}}(1^\lambda, 1^N, \mathfrak{A}, \mathsf{KeyGen}, \Pi_{\text{SKR}}) \approx_C$$
$$\mathsf{Ideal}_{\mathcal{S},\mathcal{A}}(1^\lambda, 1^N, \mathfrak{A}, \mathsf{KeyGen}, \Pi_{\text{SKR}}).$$
The experiments are described below, where we denote the set of corrupted parties at any point as C and the set of honest parties as $H = [N] \setminus C$. At any point, the adaptive adversary may corrupt an honest party and then gets full control of that party and its secret state.

$\underline{\mathsf{Real}_{\mathcal{A}}(1^\lambda, 1^N, \mathfrak{A}, \mathsf{KeyGen}, \Pi_{\text{SKR}}):}$

- Initialize $C := \emptyset$ and a list $L := \emptyset$.
- Run $(\vec{\mathsf{pk}}_{[N]}, \vec{\mathsf{sk}}_{[N]}) \leftarrow \Pi_{\text{Init}}(1^\lambda, 1^N)$ and give $\vec{\mathsf{pk}}_{[N]}$ to the adversary with oracle access: $\mathcal{A}^{O_{\text{Cor}}(\cdot), O_{\text{Back}}(\cdot), O_{\text{Rec}}(\cdot), O_{\text{RO}}(\cdot)}$.
- When $\mathcal{A}$ returns an output $\mathsf{Out}_{\mathcal{A}}$, output $(\mathsf{Out}_{\mathcal{A}}, \mathsf{Out}_{\mathcal{H}})$ where $\mathsf{Out}_{\mathcal{H}}$ is the output of all honest parties. Each honest party's output is undefined, until that party initiates Π_{Rec}. If that execution succeeds then the output is defined to be the recovered secret key, sk_i, otherwise it is defined to be $\perp$. If no such session is ever initiated, then the output stays undefined at the end.

[6]For example the generated public key can serve as an identity. Without loss of generality we denote the identity of P_i simply by an integer-i.

[7]It is also possible that a party sets either or both of the keys to $\perp$. In our protocol, however, we consider each party executes a standard KeyGen, e.g. for specific signature schemes.

$\underline{\mathsf{Ideal}_{\mathcal{A},\mathcal{A}}(1^\lambda, 1^N, \mathfrak{A}, \mathsf{KeyGen}, \Pi_{\mathsf{SKR}})}$:

- Initialize $C := \emptyset$ and a list $L := \emptyset$.
- Run $(\vec{\mathsf{pk}}_{[N]}, \vec{\mathsf{sk}}_{[N]}) \leftarrow \Pi_{\mathsf{Init}}(1^\lambda, 1^N)$ and give $\vec{\mathsf{pk}}_{[N]}$ to the adversary with oracle access: $\mathcal{A}^{\mathcal{S}_{\mathsf{Cor}}(\cdot), \mathcal{S}_{\mathsf{Back}}(\cdot), \mathcal{S}_{\mathsf{Rec}}(\cdot), \mathcal{S}_{\mathsf{RO}}(\cdot)}$.
- When $\mathcal{A}$ returns an output $\mathsf{Out}_{\mathcal{A}}$, output $(\mathsf{Out}_{\mathcal{A}}, \mathsf{Out}_{\mathcal{H}})$ and stop, where $\mathsf{Out}_{\mathcal{H}}$ is defined below within $\mathcal{S}_{\mathsf{Rec}}$.

Here O_{RO} is a random oracle and $\mathcal{S}_{\mathsf{RO}}$ is the random oracle simulated by the simulator. The rest of the oracles are defined below, where we highlight the part where it differs in case of simulator by blue.

$\underline{O_{\mathsf{Cor}}(C')/\mathcal{S}_{\mathsf{Cor}}(C')}$:

- Update $C := C \cup C'$. Run $\mathsf{CorCheck}_{\mathfrak{A}}(L, C)$, if it returns 1, then abort.
- If not, then for each $i \in C'$, give the secret state of party P_i to $\mathcal{A}$. For $\mathcal{S}_{\mathsf{Cor}}$, instead give $\vec{\mathsf{sk}}_{C'}$ to $\mathcal{S}$, who sends a simulated secret state to $\mathcal{A}$.

$\underline{O_{\mathsf{Back}}(i, B)/\mathcal{S}_{\mathsf{Back}}(i, B)}$:

- If there is an entry $(i, *, *) \in L$, then skip.
- Else run $\mathsf{pub} \leftarrow \Pi_{\mathsf{Back}}(i, B, \vec{\mathsf{pk}}_B, \vec{\mathsf{sk}}_{B\cup\{i\}})$ using the honest parties P_H and allowing $\mathcal{A}$ to control the corrupt parties P_C. For $\mathcal{S}_{\mathsf{Back}}$, let $\mathcal{S}$ control the honest parties without $\vec{\mathsf{sk}}_H$ – $\mathcal{A}$ still controls the corrupt parties.
- Then store (i, B, pub) into L.
- Run $\mathsf{CorCheck}_{\mathfrak{A}}(L, C)$, if it returns 1, then abort.

$\underline{O_{\mathsf{Rec}}(i, R)/\mathcal{S}_{\mathsf{Rec}}(i, R)}$:

- If there is no entry $(i, *, *)$ in L, then skip.
- Else retrieve (i, B, pub) from L and run $\Pi_{\mathsf{Rec}}(i, R, \vec{\mathsf{pk}}_R, \mathsf{pub}, \vec{\mathsf{sk}}_R)$ using the honest parties P_H and allowing $\mathcal{A}$ to control the corrupt parties P_C. For $\mathcal{S}_{\mathsf{Rec}}$, let $\mathcal{S}$ control the honest parties without $\vec{\mathsf{sk}}_H$ – $\mathcal{A}$ still controls the corrupt parties. Also, the honest party's output vector $\mathsf{Out}_{\mathcal{H}}$ is defined within $\mathcal{S}_{\mathsf{Rec}}$.

The following predicate $\mathsf{CorCheck}_{\mathfrak{A}}$ checks whether it is possible for the adversary to recover an honest party's key trivially through corrupting more than what is allowed.

$\underline{\mathsf{CorCheck}_{\mathfrak{A}}(L, C)}$:

- If there exists an $(i, B, \mathsf{pub}) \in L$ and $R \subseteq C$ for which $i \in H$ and $(B, R) \in \mathfrak{A}$ then return 1. Otherwise, return 0.

Remark 4.1. *Here for simplicity we assume that the backup protocol is executed only once by each party. In reality it can be executed multiple times. Our definition can be extended to that easily. Looking ahead, our construction (given in Section 6), would need some simple changes to make it work in that setting. In particular, a unique session id, which, in combination with the party id, should be used in place of only party id.*

Remark 4.2. *We stress that our protocol captures security with abort, in that a malicious adversary (in our case, can also be adaptive) can define the honest party's output to be abort. This is captured formally by our definition by including all honest party's outputs in the variable* $\mathsf{Out}_{\mathcal{H}}$*. At the beginning each honest party's output stays undefined. For every honest party-i, that initiates a recovery protocol* Π_{Rec}*, if a recovery succeeds, then a correct output, namely* sk_i *is included into* $\mathsf{Out}_{\mathcal{H}}$*. If the recovery fails, then the output is defined to be* $\perp$*. The simulated oracle* $\mathcal{S}_{Rec}$ *handles this case in the proof.*

5 BOTTOM-UP SECRET SHARING

Consider a set of N parties $P_1, \dots, P_N$, where each party P_i has an established identity denoted simply by $i \in [N]$. We formalize a secret-sharing scheme for a $(t+1)$-out-of-$(n-1)$ access structure, (first used in [3] 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 of the secret.

Definition 5.1 (Bottom-Up Secret Sharing (BUSS)). For $n, t, N \in \mathbb{N}$ such that $t + 1 \le n - 1 < N$, a bottom-up secret sharing scheme for a $(t+1)$-out-of-$(n-1)$ threshold access structure over the set $[N]$ consists of algorithms with the following syntax:

- $\mathsf{Share}(s, \vec{\sigma}_B, B)$: Takes as input a secret s, $(n-1)$ shares $\vec{\sigma}_B$, and the corresponding set of indices $B \subseteq [N] \setminus \{i\}$ such that $|B| = n - 1$. Then Share outputs a public value φ.
- $\mathsf{Recon}(\varphi, \vec{\sigma}_R, R)$: Takes as input the public value φ, $(t+1)$ shares $\vec{\sigma}_R$, and the corresponding set of indices $R \subseteq [N]$ such that $|R| = t + 1$. Then Recon outputs a secret s or outputs $\perp$ if the procedure fails.

Furthermore, the BUSS scheme satisfies the following notions of correctness and perfect adaptive simulation security.

Correctness. For any secret s, any sets R, B such that $R \subseteq B \subset [N]$, $|R| = (t+1)$, $|B| = (n-1)$, and any $\vec{\sigma}_B$:

$$\Pr\left[\, s \leftarrow \mathsf{Recon}(\varphi, \vec{\sigma}_R, R) \,\middle|\, \varphi \leftarrow \mathsf{Share}(s, \vec{\sigma}_B, B) \,\right] = 1$$

Perfect Adaptive Simulation Security. For any unbounded adversary $\mathcal{A}$, there exists a pair of algorithms $(\mathsf{SimShare}, \mathsf{SimComb})$ such that the following two distributions are identical.

- $\mathsf{RealSh}_{\mathcal{A}}$:
  - Receive $(s, \vec{\sigma}_C, C, B)$ from $\mathcal{A}$, such that $C \subseteq B$, $|C| \le t$, and $|B| = (n-1)$.
  - Sample $\vec{\sigma}_{B\setminus C}$ uniformly at random.
  - Run $\varphi \leftarrow \mathsf{Share}(s, \vec{\sigma}_B, B)$, and give φ to $\mathcal{A}$.
  - When $\mathcal{A}$ queries (Share, C') for $C' \subseteq B \setminus C$, update $C := C \cup C'$. If $|C| > t$, then abort. If not, then give $\vec{\sigma}_{C'}$ to $\mathcal{A}$.
  - Finally give $\vec{\sigma}_{B\setminus C}$.
  - Output whatever $\mathcal{A}$ returns.
- $\mathsf{IdealSh}_{\mathcal{A}}$:
  - Receive $(s, \vec{\sigma}_C, C, B)$ from $\mathcal{A}$, such that $C \subseteq B$, $|C| \le t$, and $|B| = (n-1)$.
  - Run $\varphi, \vec{\tau} \leftarrow \mathsf{SimShare}(C)$ and give φ to $\mathcal{A}$.
  - When $\mathcal{A}$ queries (Share, C') for $C' \subseteq B \setminus C$, update $C := C \cup C'$ and $\vec{\sigma}_{C'} = \vec{\tau}_{C'}$. If $|C| > t$, then abort. If not, then give $\vec{\sigma}_{C'}$ to $\mathcal{A}$.
  - Finally compute $\vec{\sigma}_{B\setminus C} \leftarrow \mathsf{SimComb}(s, B, C, \vec{\sigma}_C, \varphi)$ and give $\vec{\sigma}_{B\setminus C}$ to $\mathcal{A}$.
  - Output whatever $\mathcal{A}$ returns.

The notion of perfect adaptive simulation security says morally that the real-world φ does not reveal any information about s or any party's shares to the adversary. Note that in $\mathsf{IdealSh}_{\mathcal{A}}$, SimShare computes the public value φ given only C as input, so φ does not depend on s or on any party's shares.

Furthermore, we allow the adversary to adaptively corrupt parties by making queries of the form (Share, C'). On such a query, the adversary learns the shares $\vec{\sigma}_{C'}$ as long as the total number of parties that they've corrupted remains $\le t$.

5.1 Constructing BUSS

We construct a simple BUSS scheme for a $(t+1)$-out-of-$(n-1)$ threshold access structure over a finite field $\mathbb{F}$ 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 $\varphi := \big(f(-1), \dots, f\big(-(n-t-1)\big)\big)$.
- $\mathsf{Recon}(\varphi, \vec{\sigma}_R, R)$:
  Parse φ as $\big(f(-1), \dots, f\big(-(n-t-1)\big)\big)$. Concatenate these $(n-t-1)$ points with the following $|R| = t+1$ points: $\{f(j) := \sigma_j\}_{j \in R}$. Then use Lagrange interpolation to compute the unique polynomial f of degree $n-1$ that passes through those n points. Finally output $s := f(0)$.

Theorem 5.1 (BUSS Construction). *The above construction is a bottom-up secret sharing scheme according to Definition 5.1.*

Proof. We show correctness and perfect adaptive simulation security separately.

Correctness: The polynomial f that is computed by $\mathsf{Share}(s, \vec{\sigma}_B, B)$ has degree $(n-1)$, so it is uniquely determined by any n points on the polynomial. Next, Recon knows the value of $f(x)$ for every $x \in \{-1, \dots, -(n-t-1)\} \cup R$. This constitutes n points on the polynomial because $|R| = t+1$. Then Recon will reconstruct the polynomial f that was computed by Share, and it will compute the correct value $s = f(0)$.

Perfect Adaptive Simulation Security: Let us first construct $\mathsf{SimShare}(C)$ and $\mathsf{SimComb}(s, B, C, \vec{\sigma}_C, \varphi)$ as follows:

- $\mathsf{SimShare}(C)$: Sample $\varphi \xleftarrow{\$} \mathbb{F}^{n-t-1}$ and $\vec{\tau} \xleftarrow{\$} \mathbb{F}^{n-1}$, and output $(\varphi, \vec{\tau})$.
- $\mathsf{SimComb}(s, B, C, \vec{\sigma}_C, \varphi)$:
  (1) Partition $H = B \setminus C$ into two disjoint sets H' and H'' such that $|H'| = t - |C|$. [8]
  (2) Sample $\vec{\sigma}_{H'} \xleftarrow{\$} \mathbb{F}^{t-|C|}$.
  (3) Use $(s, \vec{\sigma}_C, \vec{\sigma}_{H'}, \varphi)$ to define n points:
  $$(0, s), (j, \sigma_j)_{j \in C \cup H'}, (-1, \varphi_1), \dots, (-(n-t-1), \varphi_{n-t-1})$$
  Interpolate the unique polynomial f of degree $n-1$ that passes through these points.
  (4) For each $j \in H''$, compute $\sigma_j = f(j)$. Then output $\vec{\sigma}_H = \vec{\sigma}_{H'} \cup \vec{\sigma}_{H''}$.

Now, we prove the following claim.

Claim 5.1. *The outputs of* $\mathsf{RealSh}_{\mathcal{A}}$ *and* $\mathsf{IdealSh}_{\mathcal{A}}$ *are identically distributed.*

Proof. Let C_0 be the value of C at the start of the game ($\mathsf{RealSh}_{\mathcal{A}}$ or $\mathsf{IdealSh}_{\mathcal{A}}$), and let C_1 be the value of C at the end, after the adversary's Share queries. Let us partition $B \backslash C_1$ into H' and H'' such that $|H'| = t - |C_1|$. This is possible assuming that $|C_1| \le t$. Note that B can be partitioned into $(C_0, C_1 \backslash C_0, H', H'')$.

The variables $(s, \vec{\sigma}_{C_1}, \vec{\sigma}_{H'}, \varphi)$ together define n points on the polynomial f. Since f has degree $(n-1)$, these n points uniquely determine f. Then for any $(s, \vec{\sigma}_{C_1}, \vec{\sigma}_{H'})$ and φ, there is a unique $\vec{\sigma}_{H''}$ such that $\varphi = \mathsf{Share}(s, (\vec{\sigma}_{C_1} \cup \vec{\sigma}_{H'} \cup \vec{\sigma}_{H''}), B)$.

Furthermore, $(s, \vec{\sigma}_{C_1}, \vec{\sigma}_{H'}, \vec{\sigma}_{H''})$ also define n points on f. So for any $(s, \vec{\sigma}_{C_1}, \vec{\sigma}_{H'})$ and $\vec{\sigma}_{H''}$, there is a unique φ such that $\varphi = \mathsf{Share}(s, (\vec{\sigma}_{C_1} \cup \vec{\sigma}_{H'} \cup \vec{\sigma}_{H''}), B)$.

In $\mathsf{RealSh}_{\mathcal{A}}$, $\vec{\sigma}_{B \backslash C_0}$ is sampled uniformly at random. Then φ is uniformly random due to the randomness of $\vec{\sigma}_{H''}$ which is hidden from the adversary. Furthermore, on each query (Share, C'), $\mathcal{A}$ receives freshly random shares $\vec{\sigma}_{C'}$ that are independent of their view so far, or they receive $\perp$. Finally, $\mathcal{A}$ receives $\vec{\sigma}_{H'}$, which is uniformly random and independent of $\mathcal{A}$'s view so far, and $\vec{\sigma}_{H''}$, which is the unique value for which $\varphi = \mathsf{Share}(s, (\vec{\sigma}_{C_1} \cup \vec{\sigma}_{H'} \cup \vec{\sigma}_{H''}), B)$.

In $\mathsf{IdealSh}_{\mathcal{A}}$, φ is sampled uniformly at random by SimShare. Then on each query (Share, C'), $\mathcal{A}$ receives $\vec{\sigma}_{C'} = \vec{\tau}_{C'}$, which was sampled (as candidate future corruption shares) independently of the adversary's view so far, by SimShare. Finally, SimComb samples $\vec{\sigma}_{H'}$ uniformly at random and chooses $\vec{\sigma}_{H''}$ to be the unique value for which $\varphi = \mathsf{Share}(s, (\vec{\sigma}_{C_1} \cup \vec{\sigma}_{H'} \cup \vec{\sigma}_{H''}), B)$.

This shows that the distribution of the values provided to $\mathcal{A}$ is the same in $\mathsf{RealSh}_{\mathcal{A}}$ and $\mathsf{IdealSh}_{\mathcal{A}}$. Then $\mathcal{A}$'s final output will be identically distributed in both hybrids. □

This concludes the proof of the theorem. □

6 OUR SOCIAL KEY RECOVERY SCHEME

Key Generation. We consider a KeyGen algorithm which generates a key-pair (sk, pk). KeyGen has the following requirements.

(1) We assume that for every pk there exists a unique sk such that (pk, sk) are a valid output of KeyGen.
(2) One can efficiently verify whether a given (pk, sk) are a valid output of KeyGen.
(3) It is hard to guess sk given pk. More formally, we say that for any PPT adversary $\mathcal{A}$,

$$\Pr\left[\mathcal{A}(1^\lambda, \mathsf{pk}) = \mathsf{sk} : (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)\right] \le \mathsf{negl}(\lambda)$$

Let this hardness assumption be known as $\mathcal{P}_{\mathsf{KeyGen}}$.

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 a 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_{cmnty} to socially recover keys for any KeyGen satisfying the above conditions in a community of N parties $P_1, \dots, P_N$ for a $(t+1)$-out-of-n threshold access structure for any $t < n \le N$.[9] We assume a hash function $\mathsf{H} : \mathbb{F} \rightarrow \mathbb{F}$, modeled as a random oracle. The construction is provided in Figure 2.

[8]This is possible because $|C| \le t$ and $t+1 \le |B|$. Then $|C \cup H'| = t$.

[9]We note that the backup set of our access structure is of size $(n-1)$, whereas the recovery set is of size $(t+1)$. However, we call this access structure $(t+1)$-out-of-n, as opposed to $(t+1)$-out-of-$(n-1)$ because also accounting for the secret, the total number of secret/share in the system is n.

- $\Pi_{\text{Init}}(1^\lambda, 1^N)$: Each party P_i:
  - Runs $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
  - Publishes pk_i and store sk_i.
- $\Pi_{\text{Back}}(i, B, \vec{\mathsf{pk}}_B, \vec{\mathsf{sk}}_{R\cup\{i\}})$: A party P_i runs this protocol as a key-owner with a set of $(n-1)$ guardians $\{P_j\}_{j\in B}$ as follows:
  - On request from P_i each guardian P_j computes $\sigma_{i,j} := \mathsf{H}(i, \mathsf{sk}_j)$ and send 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 $\mathsf{pub} := \varphi$.
- $\Pi_{\text{Rec}}(i, R, \vec{\mathsf{pk}}_{[N]}, \mathsf{pub}, \vec{\mathsf{sk}}_R)$: 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 computes $\sigma_{i,j} := \mathsf{H}(i, \mathsf{sk}_j)$ and sends that to P_i.
  - The key-owner, on receiving $(\sigma_{i,j})_{j\in R}$, retrieve $\varphi := \mathsf{pub}$ and compute $s \leftarrow \mathsf{Recon}(\varphi, (\sigma_{i,j})_{j\in R}, R)$.
  - 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 $\perp$.

Figure 2: The SKR protocol Π_{cmnty}

Remark 6.1. *As mentioned earlier, instead of i we could also use* pk_i*, and compute* $\sigma_{i,j} := H(\mathsf{pk}_i, \mathsf{sk}_j)$ *instead. This does not change anything. Also, recall that if multiple back session is executed by the same key-owner, we need a unique session id* sid*, which would be used to derive* $\sigma_{i,j} := H(\mathsf{sid}, \mathsf{pk}_i, \mathsf{sk}_j)$ *– this would ensure that the shares derived in different sessions by the same guardian are uncorrelated (as long as* sk_j *is hidden).*

Remark 6.2. *Note that, for the protocol to work, the public values* pub *just needs to be stored reliably, such that, they can be accessed unaltered whenever needed. We may use immutable bulletin boards to implement that, though it does not need to be accessed by all parties, but only the key-owner during recovery. Therefore, just a reliable storage accessible by the owner may suffice. However, in case of permanent disappearance, e.g. demise of the owner, it is important that the public value can be accessed by non-owners to recover the secret, otherwise the secret will be permanently lost – even if all* $(n-1)$ *guardians come together they can not recover the secret, as the secret polynomial in the BUSS scheme has degree* $(n-1)$.

Now we analyze our protocol formally. The access structure $\mathfrak{T}_{n,(t+1)}$ for which the scheme is secure contains all (B, R) for which $|R| = (t+1)$, $|B| = (n-1)$ and $R \subseteq B$. It immediately extends to B of arbitrary size, as long as it is a superset of R – we stick to equal sized B for simplicity of exposition.

Theorem 6.1. *Let* KeyGen *be a* standard *key-generation algorithm for which* $\mathcal{P}_{\mathsf{KeyGen}}$ *holds. Then the protocol* Π_{cmnty} *among a community of N parties* $P_1, \ldots, P_N$ *is a secure social key recovery scheme – as per Definition 4 – for access structure* $\mathfrak{T}_{n,(t+1)}$ *for any* $t+1 < n \leq N$ *against all polynomial-time malicious adaptive adversaries that corrupt* $\leq t$ *parties.*

Proof. To prove the theorem we need to show that for any adaptive and malicious adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that the adversary's interactions with the oracles in the real world are indistinguishable from its interactions with the simulated oracles in the ideal world.

We construct the simulator as follows:

The simulator $\mathcal{S}$.

- Secret states:
  - The simulator maintains a set U which contains elements of the form $(i, \mathsf{sk}_i, B, \vec{\sigma}_B, \mathsf{pub}_i)$ which is essential part of the secret state of a key owner P_i from an execution of backup protocol Π_{Back} with the adversary if, at the time of execution P_i was an honest party. There can be partial entries such as $(i, \perp, B, \vec{\sigma}_{C\cap B}, \mathsf{pub}_i)$. If P_i gets corrupted at some point, the partial entry is updated to a complete entry $(i, \mathsf{sk}_i, B, \vec{\sigma}_B, \mathsf{pub}_i)$, and is given to the adversary as part of a secret state. Note that, if P_i is corrupt during the execution of Π_{Back}, then no such entry is made to U.
  - The simulator also maintains a set V containing entries of the form $(j, \sigma_{i,j})$. This set is populated when a backup protocol is executed from an already corrupt key-owner P_i, and P_j was an honest party at that moment. Note that, elements of this set were already known to the adversary as transcripts of the backup protocol even when P_j was honest. Nevertheless, since no erasure is assumed, this should be included in the state, and would be given to the adversary on corruption to make the simulation accurate.
  - The entire simulated secret state of any party P_i consists of $(i, \mathsf{sk}_i, B, \vec{\sigma}_B, \mathsf{pub}_i)$ and $\{(i, \sigma_{k,i})\}_k$. On an adaptive corruption P_i, $(\mathsf{sk}_i, (\vec{\sigma}_B), \{\sigma_{i,k}\}_k)$ is handed over to the adversary.
- Random Oracle:
  1. $R_{\mathsf{sim}}(i, j)$: R_{sim} is internally computed and possibly programmed by $\mathcal{S}$ and accepts queries of the form $(i, j) \in [N] \times [N]$. Upon receiving query (i, j), check if (i, j) has previously been queried to R_{sim}. If so, then give the same response as the previous time. If not, then sample $y \xleftarrow{\$} \mathbb{F}$, store the equation $R_{\mathsf{sim}}(i, j) = y$, and respond with y.
  2. $\mathcal{S}_{\text{RO}}(x)$: Upon a RO query x:
     (a) Check if (1) $x = (i, \mathsf{sk})$ for some $i \in [N]$ and some key sk, and (2) $(\mathsf{sk}, \mathsf{pk}_j)$ is a valid key-pair for some $j \in [N]$. If both conditions are satisfied, then respond with $R_{\mathsf{sim}}(i, j)$.
     (b) If the conditions are not both satisfied, then check if x has been previously queried to H. If so, then give the same response as the previous time. If not, then sample $y \xleftarrow{\$} \mathbb{F}$, store the equation $H(x) = y$, and respond with y.
- The oracle $\mathcal{S}_{\text{Back}}(i, B)$ is simulated as follows:
  1. If $i \in H$, then when the adversary returns $(\sigma_{i,j})_{j\in C\cap B}$, then run $\varphi_i \leftarrow \mathsf{SimShare}(C\cap B)$ and publish φ_i as pub_i. Also append $(i, \perp, B, (\sigma_{i,j})_{j\in C\cap B}, \mathsf{pub}_i)$ in the set U.
  2. If $i \in C$, then on behalf of each honest party P_j such that $j \in H \cap B$, compute $\sigma_{i,j} := R_{\mathsf{sim}}(i, j)$, and send $\sigma_{i,j}$

as party P_j's response to the adversary. Also append $(j, \sigma_{i,j})$ to set V.

- The oracle $\mathcal{S}_{\mathsf{Cor}}(C')$ is simulated as follows. For each $i \in C'$, the simulator receives sk_i and does the following:
  1. It searches for an entry $(i, \perp, B, (\sigma_{i,j})_{j \in B \cap C_0}, \mathsf{pub}_i)$ in U, where C_0 is the value of C at the time the entry was created. If found, then:
     (a) Let $C_1 = C \backslash C_0$. For each $j \in C_1$, compute $\sigma_{i,j} = R_{\mathsf{sim}}(i, j)$.
     (b) Compute values $(\sigma_{i,j})_{j \in B \backslash C} \leftarrow \mathsf{SimComb}(\mathsf{sk}_i, B, (\sigma_{i,j})_{j \in B \cap C}, \mathsf{pub}_i)$. For each $j \in B \setminus C$, program $R_{\mathsf{sim}}(i, j) := \sigma_{i,j}$.
     (c) Update the entry in U as $(i, \mathsf{sk}_i, B, (\sigma_{i,j})_{j \in B}, \mathsf{pub}_i)$. Finally, send $(\mathsf{sk}_i, (\sigma_{i,j})_{j \in B}, \{\sigma_{i,k}\}_k)$ to $\mathcal{A}$ as the secret state, where $\{\sigma_{i,k}\}_k$ are all entries (if no such entry is found, skip this) $\{(i, \sigma_{i,k})\}$ with the same i, retrieved from V.
  2. If such an entries are not found in U or V, then send sk_i as the only secret state.
- The oracle $\mathcal{S}_{\mathsf{Rec}}(i, R)$ is simulated as follows.
  1. If $i \in H$, then no non-trivial information is sent to the adversary. However, based on whether the recovery succeeds or not it needs to define i's output within $\mathsf{Out}_{\mathcal{H}}$. Therefore, it receives the responses $\sigma_{i,j}$ for each corrupt $j \in R \cap C$ from the adversary. For each honest guardian $j \in R \cap H$, compute $\sigma_{i,j} = R_{\mathsf{sim}}(i, j)$. Then it recovers $s \leftarrow \mathsf{Recon}(\varphi_i, \vec{\sigma}_R, R)$ where φ_i is obtained from U. It verifies whether s is consistent with pk_i. If yes, then define the output of party-i with $\mathsf{Out}_{\mathcal{H}}$ to be s, otherwise define it to be $\perp$.
     simulation is trivial, as nothing is sent to the adversary (because the final output is private).
  2. If $i \in C$, then for each honest guardian $j \in R \cap H$, compute $\sigma_{i,j} = R_{\mathsf{sim}}(i, j)$ and send $\sigma_{i,j}$ to P_i.

Now for any given honest party $j \in [H]$, let us define a bad event, $\mathcal{E}^j_{\mathsf{Bad}}$, to be when $\mathcal{A}$ queries $\mathcal{S}_{\mathsf{RO}}$ on (i, sk_j) for any $i \in C$. Next, let $\mathcal{E}_{\mathsf{Bad}} = \cup_j \mathcal{E}^j_{\mathsf{Bad}}$. We argue that this happens with negligible probability in the lemma stated below.

Lemma 6.1. *For any PPT adversary $\mathcal{A}$, the probability of $\mathcal{E}_{\mathsf{Bad}}$ happening is negligible in λ, as long as the problem $\mathcal{P}_{\mathsf{KeyGen}}$ is computationally hard.*

Proof. In the reduction the challenger for $\mathcal{P}_{\mathsf{KeyGen}}$ samples $(\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and gives pk^* to the reduction. The reductions simulate the ideal world experiment $\mathsf{Ideal}_{\mathcal{A}}$ with $(\mathsf{sk}^*, \mathsf{pk}^*) = (\mathsf{sk}_{j^*}, \mathsf{pk}_{j^*})$ for a uniform random $j^* \in [N]$ as follows:

1. Sample $j^* \overset{\$}{\leftarrow} [N]$.
2. Set $\mathsf{pk}_{j^*} = \mathsf{pk}^*$. Implicitly, $\mathsf{sk}_{j^*} = \mathsf{sk}^*$ because every pk has a unique sk for a *standard* KeyGen. Then sample $(\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow \mathsf{KeyGen}(1^\lambda)$ for every $j \in [N] \backslash \{j^*\}$ and send $\mathcal{A}$ (the SKR adversary) the values $\vec{\mathsf{pk}}_{[N]}$.
3. Now simulate the oracles $\mathcal{S}_{\mathsf{RO}}, \mathcal{S}_{\mathsf{Back}}, \mathcal{S}_{\mathsf{Rec}}, \mathcal{S}_{\mathsf{Cor}}$ as above in the $\mathsf{Ideal}_{\mathcal{A}}(1^\lambda, 1^N, \mathfrak{A}, \mathsf{KeyGen}, \Pi_{\mathsf{SKR}})$ with the following conditions:
   - If $\mathcal{E}^j_{\mathsf{Bad}}$ happens but $j \neq j^*$ then abort. Call this event $\mathcal{E}_{\mathsf{Abort}}$.
   - Else when $\mathcal{E}^{j^*}_{\mathsf{Bad}}$ happens ($\mathcal{A}$ queries $\mathcal{S}_{\mathsf{RO}}(i, \mathsf{sk}_{j^*})$), then send sk_{j^*} to the $\mathcal{P}_{\mathsf{KeyGen}}$-challenger as the answer. Call this event $\mathcal{E}_{\mathsf{Break}}$.

Now, clearly $\Pr[\mathcal{E}_{\mathsf{Break}}] \geq \Pr[\mathcal{E}^{j^*}_{\mathsf{Bad}}] \geq (1/N) \cdot \Pr[\mathcal{E}_{\mathsf{Bad}}]$. $\Pr[\mathcal{E}_{\mathsf{Bad}}]$ must be negligible because $\Pr[\mathcal{E}_{\mathsf{Break}}]$ is negligible.

□

There are two differences between Real and Ideal from $\mathcal{A}$'s point of view. (1) In Ideal, $\mathcal{S}$ implements the random oracle using $\mathcal{S}_{\mathsf{RO}}$ and R_{sim}. The second difference (2) appears when P_i is corrupted after P_i serves as the key owner in an execution of Π_{Back}. In Real, the values $(\sigma_{i,j})_{j \in B \backslash C}$ are chosen randomly, and ϕ_i is computed as $\phi_i = \mathsf{Share}(s, (\sigma_{i,j})_{j \in [B]})$. In Ideal, ϕ_i is computed by $\mathsf{SimShare}(C \cap B)$, and $(\sigma_{i,j})_{j \in B \backslash C}$ is computed by $\mathsf{SimComb}(\mathsf{sk}_i, (\sigma_{i,j})_{j \in B \cap C}, \mathsf{pub}_i)$.

(1): $\mathcal{S}$ simulates the random oracle correctly if $\mathcal{E}_{\mathsf{Bad}}$ does not occur. For every distinct query that $\mathcal{A}$ makes to $\mathcal{S}_{\mathsf{RO}}$, $\mathcal{S}$ samples the response uniformly and independently at random. Furthermore, if $\mathcal{A}$ queries $\mathcal{S}_{\mathsf{RO}}(i, \mathsf{sk}_j)$ for some $j \in [N]$, the output will be the same as $\sigma_{i,j}$, which $\mathcal{S}$ provides on behalf of P_j whenever P_j serves as a guardian for P_i. Finally, in $\mathcal{S}_{\mathsf{Cor}}$, $\mathcal{S}$ programs $R_{\mathsf{sim}}(i, j)$ to $\sigma_{i,j}$ for every $j \in B \backslash C$. The adversary has not yet queried R_{sim} on (i, j), assuming that $\mathcal{E}_{\mathsf{Bad}}$ does not happen. So the adversary's view is the same as if $R_{\mathsf{sim}}(i, j)$ were set to a random initial value and were never reprogrammed.

(2): The distribution of $((\sigma_{i,j})_{j \in B \backslash C}, \phi_i)$ is the same in Real and Ideal due to the perfect adaptive simulation security of the secret sharing scheme. The problem of distinguishing Real and Ideal exactly corresponds to the problem of distinguishing RealSh and IdealSh.

This concludes the proof.

□

7 EVALUATION

We present our empirical evaluations in this section.

7.1 Implementation and Setup

We implement our scheme in Rust utilizing the RustCrypto's elliptic-curves [13] library for cryptographic operations and tokio [27] for network communication with ~ 2572 lines of code. We conduct our experiments on Google Cloud Platform (GCP) with each entity realized as a separate GCP N2D instance, with a 2.25 GHz AMD EPYC CPU and 16 GB of RAM. We evaluate the performance of our scheme in both LAN and WAN settings:

LAN setting: The GCP N2D instances for all parties are co-located within the same geographical region, achieving an average bandwidth of 880.64 MB/s and a network latency of 1.3 ms.

WAN setting: The instances for all parties are geo-distributed. This configuration yields an average bandwidth of 18.5 MB/s and a network latency of 138.0 ms.

Each data point presented in our results represents the average of 10 runs. We assess the performance of our system using four different combinations of elliptic curves and hash functions. To evaluate the scalability of our scheme, we conduct experiments by varying the number of parties.

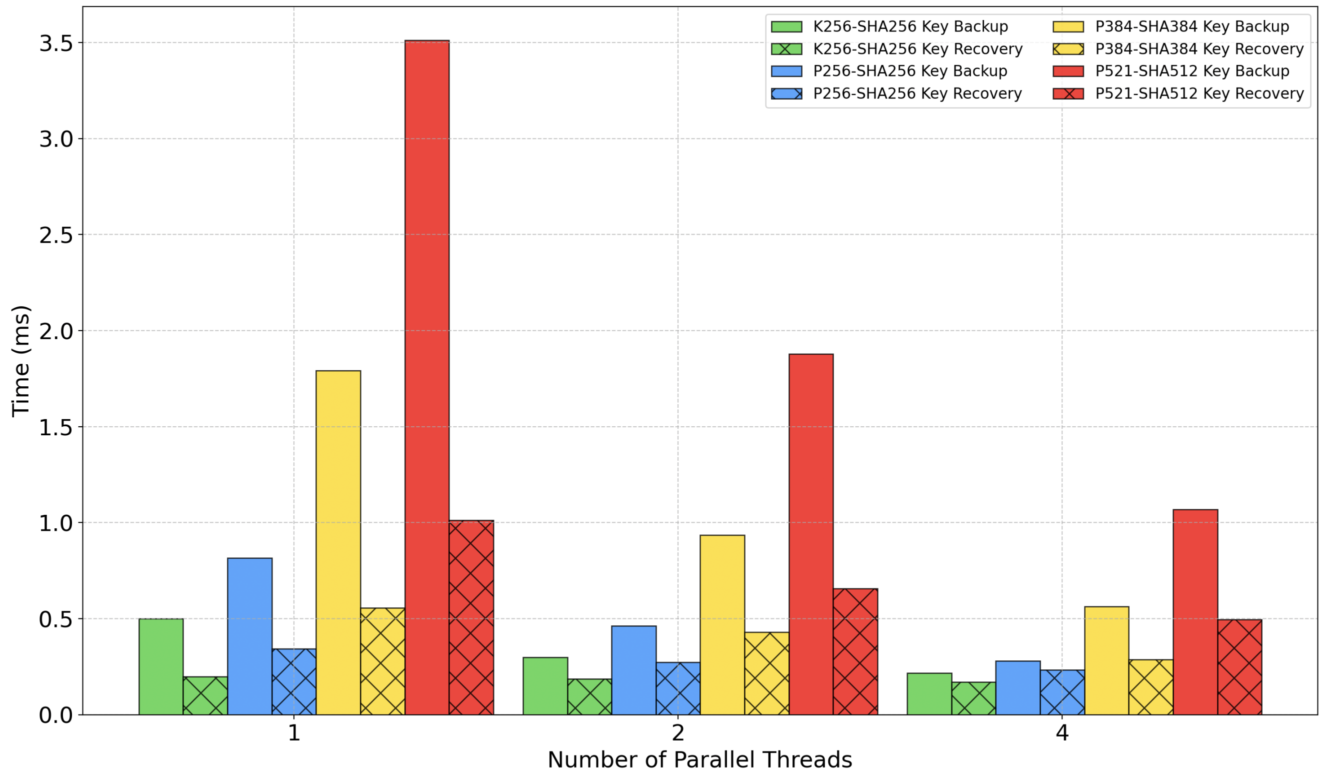

Figure 3: Compute cost comparison for creating a key backup and performing a key recovery for $n = 7$ ($n - 1$ guardians), considering {1, 2, 4} parallel threads.

7.2 Experimental Results

In this section, we evaluate the performance of our social key recovery scheme for key backup and recovery operations. Figure 3 illustrates the computation cost (excluding network time) for creating a key backup and performing a key recovery for $n = 7$ parties where $t = 3$. The key-owner requires 0.5 - 3.5 ms to create a key backup, depending on the selected elliptic curve and hash function. With 4 parallel threads, the key backup cost is reduced to 0.2 - 1.1 ms. In the event of key loss, the key-owner can perform the key recovery operation in 0.2 - 1.0 ms using a single thread, or 0.1 - 0.5 ms with 4 parallel threads.

We also evaluate the end-to-end performance of our key recovery scheme by running the key-owner and each guardian on separate GCP instances in both LAN and WAN settings, with point-to-point communication channels between the key-owner and each guardian. Table 1 presents the total time for key backup and recovery for different choices of elliptic curves and hash functions. As expected, the time for both key backup and recovery operations scales linearly with the number of parties. For instance, using K256-SHA256 in the LAN setting, as n increases from 3 to 11, the key backup time rises linearly from 1.839 ms to 5.696 ms and the key recovery time increases linearly from 0.83 ms to 1.347 ms. Similarly, using P521-SHA512 in the WAN setting, the key backup time increases from 146.747 ms to 149.670 ms and the key recovery time increases from 146.625 ms to 148.371 ms as n varies from 3 to 11. This demonstrates the scalability of our protocol in both regional (LAN) and geo-distributed (WAN) settings.

8 EXTENSIONS

In this section we discuss a number of extensions for our concrete construction (our generic SKR protocol Π, given in Figure 2 instantiated with our BUSS scheme from Section 5.1) that maybe useful in practice. We choose not to formalize this to keep the core ideas simple.

Key Update. A natural question to consider is what happens when a guardian performs a key update. To accommodate that, we describe a simple mechanism to which can be easily incorporated into our existing protocol as follows:

- Let us assume the old key of a guardian with identity-j is sk_j, and the updated key is sk'_j. For each key-owner with identity-i, guardian-j computes the difference in the share $\Delta_{i,j} := \mathsf{H}(i, \mathsf{sk}_j) - \mathsf{H}(i, \mathsf{sk}'_j)$ and send that to owner-i.
- The owner implicitly defines the updated secret polynomial from the BUSS scheme $f'_i(x) := f_i(x) + \Delta_{i,j} L_{j,B}(x)$. where $L_{j,B}(x)$ is the Lagrange polynomial corresponding to point j and set B.[10]
- Update the public values, such as $f'_i(-1) = f_i(-1) + \Delta_{i,j} L_{j,B}(-1)$ and so on.

[10]Recall that, for each point $j' \in B \setminus \{j\}$, $L_{j,B}(j') = 0$ and at point j: $L_{j,B}(j) = 1$.

EC-Hash	n	LAN		WAN	
		Backup	Recovery	Backup	Recovery
K256-SHA256	3	1.839	0.830	138.866	138.709
	5	2.718	0.879	142.550	142.476
	7	3.874	0.928	146.499	146.412
	9	4.981	1.284	146.581	146.381
	11	5.696	1.347	147.250	146.971
P256-SHA256	3	1.905	0.805	142.959	142.848
	5	2.962	0.893	146.510	146.342
	7	3.671	0.978	146.897	146.634
	9	4.864	1.133	146.930	146.379
	11	5.941	1.220	147.369	146.844
P384-SHA384	3	2.182	1.158	143.296	143.248
	5	3.408	1.164	147.041	146.685
	7	4.361	1.182	147.325	146.952
	9	5.184	1.367	147.930	147.175
	11	6.480	1.454	148.411	147.360
P521-SHA512	3	2.110	1.165	146.747	146.625
	5	3.120	1.230	148.209	146.955
	7	4.448	1.377	148.560	147.477
	9	5.962	1.440	149.151	147.890
	11	7.215	1.525	149.670	148.371

Table 1: End-to-end time in milliseconds for performing a key backup and recovery for $n = 2t + 1$ ($n - 1$ guardians) in both LAN and WAN settings.

Clearly, the new share of guardian-j is $f'_i(j) = f_i(j) + \Delta_{i,j}$, and for all other guardians $j' \in B \setminus \{i\}$ we have $f'_i(j') = f_i(j')$, that means the share remains unchanged. So, the procedure is completely oblivious to them, as they may not be even aware of the update, which takes place only between the guardian-j and the corresponding key-owner, who needs to update only the public values. Nevertheless, guardian-j must contact all key-owners he is supporting. Security wise, the key-owner knows $\Delta_{i,j}$ which is not a sensitive information.

Risk of Exposure. A practical concern with our approach might be that, since each guardian needs to use their secret key for computing each share, that makes their own key exposed in the memory for a longer period. One simple way to mitigate this would be to use another key just for this purpose. While this requires additional storage, unlike our solution, still the requirement for total secret storage is constant and independent of the number of key-owners it supports. Note that, this will also eliminate the possibility of so-called "domino effect" discussed in Section 1.3.

Cold wallets. While our solution is general and is also agnostic of whether a hot or cold wallet is being used, in practice it is easier to deploy in a hot wallet setting, where secret keys can directly used to compute the shares. Cold wallets, instead never expose the key to the memory, and instead just send out the computed signatures for authorizing transactions directly. For such settings, computing shares by hashing the secret keys would not work immediately, as cold wallets are programmed to compute specific functions (such as signatures). However, since most cold wallets (e.g. Ledger [15], Trezor [29] etc.) implement RFC6979 specs [21], which specifies deterministic signatures, we can make it work with our approach as follows:

- The guardian-j generates a deterministic signature $\zeta_{i,j}$ on a specific message (Rec, i, j) o help backup/recover the key of owner i.
- Then guardian-j participates in the SKR protocol (Fig. 2) using $\sigma_{i,j} := \mathrm{H}(\zeta_{i,j})$, where H is a random oracle that maps signatures to field elements.

It works because instead of the secret keys themselves it suffices to use the deterministic signatures output by the cold-wallet on the unique message. Each $\sigma_{i,j}$ is guaranteed to be random and unpredictable due to the unforgeability guarantees of the cold-wallet. Nonetheless, we remark that if a cold wallet implements a randomized signature then this will not work, because the wallet would yield different $\zeta_{i,j}$ values during backup and recovery, which is undesirable. Further note that, this issue does not arise if the above two-key approach is used.

9 CONCLUSION

We propose a simple and efficient protocol for socially backup and recover keys within a community where each party holds a secret key (and has a corresponding public key). Our solution uses a newly introduced technique, that we formalize here as bottom-up secret sharing, as the main underlying tool. Our design enables mutual key backups within the community - where each party can act as a backup guardian for multiple key owners, and no guardian needs to store anything extra other than its own secret key.

We analyze our protocol in a formal framework, we put forward in this work. While our framework is general enough, it does not capture the so-called domino effect (Section 1.3). We leave formalizing that with a plausibly extended framework as an interesting future work.

To the best of our knowledge, ours is the first effort to formalize social recovery protocols. Additionally the mutual key back-up setting within a community of key holders has never been considered earlier. Our protocol is very simple and also practically efficient, as demonstrated by our empirical results. The new concept of bottom-up secret sharing may be of independent interest.

REFERENCES

[1] Aptos. 2024. Aptos Keyless. https://aptos.dev/en/build/guides/aptos-keyless.

[2] Argent. 2025. How to Recover My Wallet with Guardians On-Chain (Complete Guide). https://support.argent.xyz/hc/en-us/articles/360007338877-How-to-recover-my-wallet-with-guardians-on-chain-complete-guide. Accessed: 2025-01-09.

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

[4] Mihir Bellare, Wei Dai, and Phillip Rogaway. 2020. Reimagining secret sharing: Creating a safer and more versatile primitive by adding authenticity, correcting errors, and reducing randomness requirements. *Proceedings on Privacy Enhancing Technologies* (2020).

[5] BitGo. 2025. Wallet Recovery Guide. https://developers.bitgo.com/guides/wallets/recover. Accessed: 2025-01-09.

[6] Sam Blackshear, Konstantinos Chalkias, Panagiotis Chatzigiannis, Riyaz Faizullabhoy, Irakliy Khaburzaniya, Eleftherios Kokoris-Kogias, Joshua Lind, David Wong, and Tim Zakian. 2021. Reactive Key-Loss Protection in Blockchains. In *Financial Cryptography and Data Security. FC 2021 International Workshops - CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 12676)*, Matthew Bernhard, Andrea

Bracciali, Lewis Gudgeon, Thomas Haines, Ariah Klages-Mundt, Shin'ichiro Matsuo, Daniel Perez, Massimiliano Sala, and Sam Werner (Eds.). Springer, 431–450.

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. 514–532. https://doi.org/10.1007/3-540-45682-1_30

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

[9] Ran Canetti. 2000. Security and Composition of Multiparty Cryptographic Protocols. 13, 1 (Jan. 2000), 143–202. https://doi.org/10.1007/s001459910006

[10] Panagiotis Chatzigiannis, Konstantinos Chalkias, Aniket Kate, Easwar Vivek Mangipudi, Mohsen Minaei, and Mainack Mondal. 2023. SoK: Web3 Recovery Mechanisms. *IACR Cryptol. ePrint Arch.* (2023), 1575. https://eprint.iacr.org/2023/1575

[11] Cypherock. 2025. Cypherock Documentation. https://docs.cypherock.com/. Accessed: 2025-01-09.

[12] Poulami Das, Sebastian Faust, and Julian Loss. 2019. A Formal Treatment of Deterministic Wallets. 651–668. https://doi.org/10.1145/3319535.3354236

[13] RustCrypto Developers. 2025. Elliptic Curves Implementation. https://github.com/RustCrypto/elliptic-curves. Accessed: 2025-01-09.

[14] Dash Core Group. 2025. Dash: A Privacy-Centric Cryptocurrency. https://www.exodus.com/assets/docs/dash-whitepaper.pdf. Accessed: 2025-01-09.

[15] Ledger. [n. d.]. Ledger: Hardware Wallet for Cryptocurrency Security. https://www.ledger.com/. Accessed: 2025-01-08.

[16] Yehuda Lindell. 2017. How to Simulate It - A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*, Yehuda Lindell (Ed.). Springer International Publishing, 277–346. https://doi.org/10.1007/978-3-319-57048-8_6

[17] Yehuda Lindell. 2023. Cryptography and MPC in Coinbase Wallet as a Service (WaaS). https://www.coinbase.com/blog/digital-asset-management-with-mpc-whitepaper

[18] Ryan Little, Lucy Qin, and Mayank Varia. 2024. Secure Account Recovery for a Privacy-Preserving Web Service. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, Davide Balzarotti and Wenyuan Xu (Eds.). USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/little

[19] MetaMask. 2025. MetaMask Documentation. https://docs.metamask.io/. Accessed: 2025-01-09.

[20] Allan B. Pedin, Nazli Siasi, and Mohammad Sameni. 2023. Smart Contract-Based Social Recovery Wallet Management Scheme for Digital Assets. In *Proceedings of the 2023 ACM Southeast Conference* (Virtual Event, USA) *(ACM SE '23)*. Association for Computing Machinery, New York, NY, USA, 177–181. https://doi.org/10.1145/3564746.3587016

[21] T. Pornin. 2013. RFC 6979: Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA).

[22] Sequence. 2025. Sequence Documentation. https://docs.sequence.xyz/. Accessed: 2025-01-09.

[23] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[24] Sui. 2023. zkLogin. https://docs.sui.io/concepts/cryptography/zklogin.

[25] Jacob Swambo and Antoine Poinsot. 2021. Risk Framework for Bitcoin Custody Operation with the Revault Protocol. In *Financial Cryptography and Data Security. FC 2021 International Workshops*, Matthew Bernhard, Andrea Bracciali, Lewis Gudgeon, Thomas Haines, Ariah Klages-Mundt, Shin'ichiro Matsuo, Daniel Perez, Massimiliano Sala, and Sam Werner (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 3–20.

[26] New York Times. [n. d.]. 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.

[27] Tokio. 2025. Tokio: An Async Runtime for Rust. https://tokio.rs/. Accessed: 2025-01-09.

[28] Torus. 2025. Torus Documentation. https://docs.tor.us/. Accessed: 2025-01-09.

[29] Trezor. [n. d.]. Trezor: Hardware Wallet for Secure Cryptocurrency Storage. https://trezor.io/. Accessed: 2025-01-08.

[30] Zengo. 2025. Zengo: The Crypto Wallet for Everyone. https://zengo.com/. Accessed: 2025-01-09.