On Cryptocurrency Wallet Design

Ittay Eyal
Technion, Israel
IC3

Abstract

The security of cryptocurrency and decentralized blockchain-maintained assets relies on their owners safeguarding secrets, typically cryptographic keys. This applies equally to individuals keeping daily-spending amounts and to large asset management companies. Loss of keys and attackers gaining control of keys resulted in numerous losses of funds.

The security of individual keys was widely studied with practical solutions available, from mnemonic phrases to dedicated hardware. There are also techniques for securing funds by requiring combinations of multiple keys. However, to the best of our knowledge, a crucial question was never addressed: How is wallet security affected by the number of keys, their types, and how they are combined? This is the focus of this work.

We present a model where each key has certain probabilities for being *safe*, *lost*, *leaked*, or *stolen* (available only to an attacker). The number of possible wallets for a given number of keys is the Dedekind number, prohibiting an exhaustive search with many keys. Nonetheless, we bound optimal-wallet failure probabilities with an evolutionary algorithm.

We evaluate the security (complement of failure probability) of wallets based on the number and types of keys used. Our analysis covers a wide range of settings and reveals several surprises. The failure probability general trend drops exponentially with the number of keys, but has a strong dependency on its parity. In many cases, but not always, heterogeneous keys (not all with the same fault probabilities) allow for superior wallets than homogeneous keys. Nonetheless, in the case of 3 keys, the common practice of requiring any pair is optimal in many settings.

Our formulation of the problem and initial results reveal several open questions, from user studies of key fault probabilities to finding optimal wallets with very large numbers of keys. But they also have an immediate practical outcome, informing cryptocurrency users on optimal wallet design.

2012 ACM Subject Classification Security and privacy → Formal security models

Keywords and phrases cryptocurrency,wallet,key-management

1 Introduction

Cryptocurrency systems maintain their state in a database called a blockchain. The state includes user's data as well as their token balances. To change their data, and in particular to order transactions of their funds, users append data structures called *transactions* to the blockchain. To authenticate themselves, users include in the transaction a proof of their (typically pseudonymous) identity.

Unlike common centralized systems, the choice of the authentication method is at the hands of the users themselves. A user defines, with a blockchain transaction, a predicate for authentication to access her digital assets. Transaction that access those assets must include inputs to make this predicate true. The predicate is called a *wallet*, and it is specified in a dedicated programming language as a *smart contract*. Typically, wallets require one or more cryptographic signatures. The *security* of a wallet, its probability of not failing, thus relies on the safekeeping of one or more secrets – private keys maintained by the user.

In classical systems like bank accounts and credit cards, accounts are identified and users are insured against theft. They are allowed to revoke and refresh their keys using alternative authentication techniques [13]. None of those mechanisms apply to cryptocurrencies, and if a user's keys are lost or leaked, she immediately loses access to her funds. The problem

applies equally to personal wallets, holding small amounts for daily use, and to large actors such as companies holding their own[1] or their clients'[2] funds.

Previous work (§2) proposed techniques to maintain keys [5, 11], including locally-stored files, possibly encrypted, dedicated hardware [1], hosted online services, paper wallets, and brain wallets (i.e., committing the key to memory). Other work discovered techniques to implement threshold signatures [19, 15, 14], where multiple secrets are necessary to produce a signature but, unlike classical secret sharing [21], the private key is never learnt by any party. Password-protected secret sharing [2, 17] allows users to store a secret shared between multiple servers, overcoming server faults, and without the servers learning the secret without a password. Companies[3] and open-source projects[4] use wallets that require two keys, or alternatively one-out-of-two keys, two-out-of-three keys, or more elaborate schemes.

Nonetheless, to the best of our knowledge, the question of how many keys a user should maintain and what combination is the most secure was never formally explored. This is the goal of this work.

We model the system (§3) by considering a principal, called an *owner*, that wishes to secure cryptocurrency tokens. The owner can store one or more secrets called *keys*. These could be cryptographic signing keys, portions thereof, or any other secrets. If only the owner can access a key we call that key *safe*. But keys can suffer three types of *faults*. First, they can be *lost*; e.g., in a discarded hard-drive[5], a forgotten password[6], due to a fire, etc. Secondly they can be *leaked* to the adversary, e.g., by gaining access to a machine, keystroke logging, or guessing [22, 6]. Finally, they can be *stolen*; this is a combination of the previous faults, where the adversary gains access to the key and the owner loses access to it. This could happen, e.g., if the key is controlled by an online service that is compromised, or if an attacker physically steals a written secret or even a hardware key [16]. There is a certain probability for each of the keys to suffer each of the faults. Note that the four states cannot be reduced: the probability of loss and leakage can be 2% each with a theft probability of either 0 or 1%, for example. For simplicity we assume that fault probabilities of the keys are independent of one another. We also assume that the owner knows these probabilities.

To secure her funds, the owner locks her tokens in the cryptocurrency system with a smart contract that can require an arbitrary combination of the keys. For two keys it can require either key, so if one is lost she can use the other. She could also require both keys, so if one is leaked the adversary cannot move the tokens. We call the set of possible combinations defined in the contract a *wallet*. If a party has the keys for one of the combinations we say she can *satisfy* the wallet. The wallet *fails* if the owner cannot satisfy it or if an adversary can. The former happens if enough keys are either lost of stolen, and the latter if enough keys are either leaked or stolen. The owner should choose the *best wallet* (lowest failure probability) given the fault probabilities of the keys.

To analyze wallet design we use two techniques (§4). First, for wallets with up to 5 keys we are able to complete an exhaustive search of the wallet space. By enumerating the probabilities of all key states and calculating the failure probability of each wallet we can find the optimal wallet given key fault probabilities.

---

[1] `https://edition.cnn.com/2021/02/08/investing/tesla-bitcoins/index.html`
[2] `https://www.forbes.com/advisor/investing/best-crypto-exchanges`, `https://www.cmegroup.com/markets/cryptocurrencies.html`
[3] `https://keys.casa/`, `https://zengo.com/`
[4] `https://electrum.readthedocs.io/en/latest/multisig.html`
[5] `https://www.bbc.com/news/uk-wales-south-east-wales-25134289`
[6] `https://www.nytimes.com/2019/02/05/business/quadriga-cx-gerald-cotten.html`

The wallet space grows super-exponentially with the number of keys prohibiting an exhaustive search for large wallets. However, we can bound the failure probabilities of the optimal wallets using evolutionary optimization [10]. Roughly, we construct a population of wallet candidates and iteratively perturb them and select the best for the next generation.

We begin our analysis (§5) with a detailed review of the optimal wallets for up to 4 keys with specific fault probabilities. Even with two keys, we find that there are prominent cases where the owner is better off by using only one key and ignoring the other. This happens with homogeneous keys (that have the same probabilities) if the theft probability is non-negligible and the loss and leak probabilities are similar. With three keys, we confirm that the common 2/3 approach [12] is indeed the optimum for a variety of realistic settings. We illustrate why different optimal wallets are better in different settings by considering the probabilities that the owner and that the attacker can satisfy a wallet. With four keys, two distinct natural extrapolations of the 2/3 solution are optimal for a wide range of probabilities, namely either all pairs or all triads. However, in prominent cases, where theft probability is non-negligible and loss and leak probabilities are similar, the optimum is a different, asymmetric wallet.

The owner can choose how many keys to use for its wallet. Our next step is therefore to evaluate the effect of the number of keys of wallet security. As expected, the optimal-wallet failure probability drops in a general exponential trend with the number of keys. However, when the theft probability is non-negligible, there is a strong dependence on parity. For example, when all keys have the same theft probability but no other faults, failure probability does not improve when adding a key to an odd-key wallet.

Finally, the owner has a choice of which keys to use for its wallet. For example, she could use a key stored in a mobile-device and two hardware keys, or vice versa. By considering the design space for 3-key wallets, we find that it is often, but not always, best to have heterogeneous keys.

We conclude (§6) by reviewing several open directions exposed by this work. These include user studies to quantify the probabilities of key faults, dependency among key fault probabilities, and techniques to identify optimal wallets with very large numbers of keys. However, the results presented here are of immediate importance to users, who can estimate their keys' fault probabilities, choose keys wisely, and thus improve the security of their funds. We intend to open-source the tools presented here.

2 Related Work

No work we are aware of analyzes cryptocurrency wallet design, though many works discuss tools for wallet design and implement a variety of designs.

Kirstein et al. [18] assume users have two storage tiers, one that is harder to access but is more secure, and another that is easy to access but is less secure. They design a wallet contract based on Moser et al. [20] that takes advantage of the two tiers to provide both good security and good accessibility. Earlier work by Baratam [3] uses a similar technique to combine a variety of keys with different properties. He combines user-maintained keys with keys hosted by remote services and proposes a wallet contract to minimize failure probability. Both of the proposals focus on the mechanics of the wallets, but neither of them describes the key fault probabilities, analyzes the resultant wallet failure probabilities, or addresses the choice of type and number of keys.

Eskandari et al. [11] provide a taxonomy of individual key storage options, including local storage, encrypted local storage, offline storage, hosted, etc. Evaluated criteria cover security,

as well as usability and deployability. However, they do not consider multi-key wallets and overall wallet failures, only the security of individual keys.

Bonneau et al. [5] devote a whole chapter in their Systemization of Knowledge work to client-side security. In addition to per-key storage options, they also discuss the mechanics for implementing a symmetric k/n wallet, but not a formal treatment of its superior wallet security.

In their review of cryptocurrency research challenges, Barber et al. [4] also devote a chapter to client-side security. But they, too, focus on the properties of individual keys. They explicitly discuss the threat of benign key loss, but do not consider theft, and do not quantify fault probabilities and their effect.

3 Model

We explain the reasoning and simplifications resulting in our model (§3.1) followed by the formal specification (§3.2).

3.1 Rationale

A cryptocurrency user secures her assets by defining arbitrary logic to authorize access to her assets. We call this logic a *wallet*. It is implemented in a so-called smart-contract for the relevant blockchain. Once the wallet is implemented in a smart contract, assets can be allocated (sent) to it on the blockchain. When issuing commands that affect those assets, the user publishes a transaction with inputs to the smart contract, and the transaction takes effect only if the contract authorizes it.

The transaction is first published via a peer-to-peer network, and subsequently placed in the blockchain by one of its operators, called *miners*. This mechanism means that the authentication mechanism should prohibit malicious parties from being able to authenticate themselves based on observed transactions. Otherwise, miners (or really anyone) [8] could take advantage and relieve the user of her assets: They would observe a transaction, replace the order with another, and falsely authenticate.

Therefore, wallets are typically implementing by requiring cryptographic signatures matching predefined public keys. A wallet can combine any number of signature requirements, e.g., for two particular public keys, for any pair out of three options, etc. Moreover, the contract can require only a single signature, but the user partitions her secret into several parts and combines them offline before issuing a transaction. Either way, the result is the same – the wallet is secured by a set of secrets we call keys (even if they are parts of a single cryptographic key) requiring a combination of the keys to authenticate. The wallet is thus a predicate of this set of keys, and if a party has enough keys to make this predicate true we say she can *satisfy* the wallet.

We say a wallet fails if the user loses control of her assets, which could happen due to two reasons. First, the owner might lose access to keys and not be able to satisfy the wallet. Secondly, an adversary could gain access to a combination of keys that allows her to satisfy the wallet and steal the assets. This second option includes scenarios where both parties can satisfy the wallet. In practice, in such scenarios the parties can enter a bidding war, each paying more fees, trying to get the miners to place her transaction in the blockchain. The fees can be arbitrarily high, with the attacker possibly willing to pay almost the entire wallet amount to make a profit. We therefore consider the wallet failed in such scenarios.

An owner can add offline security measures to her keys. For example, she could encrypt locally-stored keys or require a PIN access code to a hardware signing device. These

approaches simply reduce the probability that the attacker gains access to a key but also increase the probability that the owner loses access [5]. They are thus covered by our model.

An owner might lose access to her keys gradually. For example, she forgets one memorized key, and after several months loses another paper key due to a basement flood. An adversary can also gain key access gradually. For example, a hacker steals a hardware signing device and subsequently retrieves a typed key with a keystroke logger as the user tries to save her assets.

In our model we flatten such series of events of keys being lost and leaked to probabilities per key. One can think of those as being evaluated at the time at which the owner wishes to transact her funds. At this point, there is a certain probability that the adversary had gained access to the keys, or that the owner had lost access.

3.2 Model Details

An *owner* maintains a set of n keys $k_1, \ldots, k_n$. Each key is in one of four states:

safe Only the user has access,
loss No one has access,
leak Both the user and the adversary have access, or
theft Only the adversary has access,

denoted $\mathcal{S}$, i.e., for each $1 \leq i \leq n$: $k_i \in \mathcal{S} = \{\mathsf{safe}, \mathsf{loss}, \mathsf{leak}, \mathsf{theft}\}$.

A scenario σ is the state vector of each of the keys, and denote by Σ_n the set of all scenarios with n keys, i.e., $\sigma \in \Sigma_n = (\mathsf{safe}, \mathsf{loss}, \mathsf{leak}, \mathsf{theft})^n$. Denote by σ_i the state of key i in scenario σ.

Denote by σ^O and σ^A the binary *availability vectors* of each of the keys in scenario σ for the owner and for the adversary, respectively. For the state of key i, σ_i, the availabilities are σ_i^O and σ_i^A. For example, if $\sigma_i = \mathsf{safe}$ then $\sigma_i^O = \mathit{True}$ and $\sigma_i^A = \mathit{False}$. Table 1 summarizes the translation from state to availability.

The states of the different keys are determined by a probability space specified by independent probabilities of each of the keys. These probabilities are described by a tuple P of probability vectors P^{safe}, P^{loss}, P^{leak} and P^{theft}. The probabilities that key i is in each of the states are denoted P_i^{safe}, P_i^{loss}, P_i^{leak}, and P_i^{theft} for safe, loss, leak, and theft, respectively. The sum of probabilities for each key i is one, $P_i^{safe} + P_i^{loss} + P_i^{leak} + P_i^{theft} = 1$.

The probability of a scenario can be calculated given the probability vector tuple. For example, for three keys, the probability that the first is safe, the second is lost and the third is leaked is $P_1^{safe} \times P_2^{loss} \times P_3^{leak}$. In general: ($\mathbb{1}_c$ is the indicator function, it equals 1 if the condition c holds and 0 otherwise)

$$\Pr[\sigma] = \prod_{i=1}^{n} \left(\sum_{s \in \mathcal{S}} \mathbb{1}_{\sigma_i = s} \times P_i^s \right) \quad . \tag{1}$$

With n keys there are 4^n possible states, each key being in one of its four possible states. We can calculate the probability of each scenario and derive the key availability probabilities for each party. Table 2 shows 5 scenarios from the $4^2 = 16$ possible with 2 keys.

A wallet w is a predicate of the availability of the n keys. For example, a wallet w that requires the availability of either keys k_1 and k_2 or keys k_2 and k_3 is defined by $w(a_1, a_2, a_3) \triangleq (a_1 \wedge a_2) \vee (a_2 \wedge a_3)$. By slight abuse of notation, for a vector $v = (v_1, v_2, v_3)$ we write $w(v)$ for $w(v_1, v_2, v_3)$.

σ_i	${\sigma^O}_i$	${\sigma^A}_i$
safe	*True*	*False*
loss	*False*	*False*
leak	*True*	*True*
theft	*False*	*True*

Table 1 Key state and availability

σ_1	σ_2	probability	σ^O	σ^A
safe	safe	$(P^{safe})^2$	(*True*, *True*)	(*False*, *False*)
safe	loss	$P^{safe} \times P^{loss}$	(*True*, *False*)	(*False*, *False*)
safe	leak	$P^{safe} \times P^{leak}$	(*True*, *True*)	(*False*, *True*)
safe	theft	$P^{safe} \times P^{theft}$	(*True*, *False*)	(*False*, *True*)
loss	safe	$P^{loss} \times P^{safe}$	(*False*, *True*)	(*False*, *False*)
...	...	...	...	...

Table 2 State enumeration

We denote by $w^{\mathsf{k/n}}$ the wallet that is satisfied by any set of k out of the n keys. The wallet w_n^{AND} is the wallet requiring all n keys ($w^{\mathsf{n/n}}$), and the wallet w_n^{OR} is the wallet requiring any of the n keys ($w^{\mathsf{1/n}}$).

A wallet w is *successful* in a scenario σ if and only if the adversary cannot satisfy it and the owner can, i.e., $w(\sigma^O) \wedge \neg w(\sigma^A)$. Otherwise, the wallet has *failed* in this scenario.

Given a probability vector tuple P, we can calculate for wallet w its success and failure probabilities, $p^P_{success}(w)$ and $p^P_{failure}(w)$, respectively. For success, we sum the probabilities of all states where the wallet is successful,

$$p^P_{success}(w) = \sum_{\sigma \in \Sigma_n} \Pr[\sigma] \times \mathbb{1}_{w(\sigma^O)} \times \mathbb{1}_{\neg w(\sigma^A)} \quad , \tag{2}$$

and the failure probability is the complement $p^P_{failure}(w) = 1 - p^P_{success}(w)$.

We denote the fact a wallet w is better than wallet w', i.e., $p^P_{failure}(w) < p^P_{failure}(w')$ by $w' \prec_P w$, omitting the subscript ($w' \prec w$) when P is clear from the context.

4 Methodology

To find the optimal wallet for a given probability vector tuple, we search the entire design space when possible (§4.1) or use an approximation (§4.2) otherwise.

4.1 Exhaustive Search

For small numbers of keys we can find the optimal wallet by exhaustively calculating the failure probabilities of all possible wallets. We observe that each wallet is a monotone boolean function – if a party can satisfy the wallet, being able to access another key will not prohibit it from satisfying the wallet. The exact number of such functions, the Dedekind number [9] minus 2 (excluding the constant functions True and False), is only known up to 8 keys [7, 23, 24]. It grows super-exponentially, making complete coverage of all possible wallets intractable. Complete coverage also becomes intractable due to the exponentially growing number of key states (4^n).

To enumerate all wallets, we express wallets as a logical *or* of sets of logical *and*'s. We call each set of *and*-ed keys a *combination*. For example, for two keys we consider four possible wallets, requiring one combination of both keys ($k_1 \wedge k_2$), requiring either of them (k_1) $\vee$ (k_2) (two combinations), but also the asymmetric options of requiring one (k_1) or the other (k_2).

We denote by $c \subset c'$ the fact that all keys in combination c are also in combination c' and say c' *covers* c. We exclude wallets where one combination covers another, e.g., ($k_1 \wedge k_2 \wedge k_3$) $\vee$ ($k_1 \wedge k_2$) as it is the same function as ($k_1 \wedge k_2$), which is evaluated.

There are a total of $(2^n - 1)$ possible non-zero combinations, and we consider all sets of those combinations that are not redundant due to one combination covering another.

Algorithm 1 Wallet enumeration

```
1 function enumerateWallets (baseWallet, prevCombi, keyCount)
2     wallets ← ∅
3     for combi = next(prevCombi) to lastCombi(n) do
4         if ∄c ∈ baseWallet : c ⊂ combi then          (Skip redundant combinations)
5             currWallet ← baseWallet ∪ {combi}
6             wallets ← wallets ∪ {currWallet}          (With new combination)
7             wallets ← wallets ∪ enumerateWallets(currWallet, combi, keyCount)
                                                        (With new combination and others)
8     return(wallets)
```

For example, with three keys there are 18 possible wallets, as follows,

$$(k_1), (k_2), (k_3), \tag{3a}$$
$$(k_1) \vee (k_2), (k_1) \vee (k_3), (k_2) \vee (k_3), \tag{3b}$$
$$(k_1) \vee (k_2) \vee (k_3), \tag{3c}$$
$$(k_1 \wedge k_2), (k_1 \wedge k_3), (k_2 \wedge k_3), \tag{3d}$$
$$(k_1 \wedge k_2) \vee (k_3), (k_1) \vee (k_2 \wedge k_3), (k_2) \vee (k_1 \wedge k_3), \tag{3e}$$
$$(k_1 \wedge k_2) \vee (k_1 \wedge k_3), (k_1 \wedge k_2) \vee (k_2 \wedge k_3), (k_1 \wedge k_3) \vee (k_2 \wedge k_3), \tag{3f}$$
$$(k_1 \wedge k_2) \vee (k_1 \wedge k_3) \vee (k_2 \wedge k_3), \tag{3g}$$
$$(k_1 \wedge k_2 \wedge k_3) \ . \tag{3h}$$

We enumerate the wallets recursively, as shown in Algorithm 1. The algorithm utilizes a lexicographic order of the combinations, implemented with the function *next* that returns the next combination in order and *last* that return the last combination, namely $k_1, \ldots, k_n$. We call the algorithm with $baseWallet = \emptyset$, $prevCombi = 0$, and the desired number of keys, n. The recursive algorithm returns the set of all wallets with all non-redundant subsequent combinations, considering redundancy with wallets in *baseWallet*.

4.2 Evolutionary Approximation

To analyze larger numbers of keys, beyond the reach of an exhaustive search, we search the space with an evolutionary algorithm [10] (Algorithm 2). We bootstrap the algorithm with a random *population* – a multiset of n_{pop} wallets. The algorithm then iteratively improves this population. In each iteration i we produce a new generation based on the population of iteration $i-1$.

We first select a multiset of n_{pop} wallets as follows. For each selection we consider $n_{selection}$ wallets uniformly at random (UAR) from iteration $i-1$ and select the best (lowest failure rate) among them. Good wallets are likely to appear multiple times in the new multiset.

We then perturb each wallet in the selection by choosing one of its combinations uniformly at random and flipping each of its key requirements with independent probability $p_{perturb}$. For example, for a three-key wallet combination of $k_1 \wedge k_2$, key k_1 is removed with probability $p_{perturb}$, key k_2 is removed with probability $p_{perturb}$, and key k_3 is added with probability $p_{perturb}$.

The resultant wallet multiset is then taken as the population of the next iteration. We repeat this until there is no improvement in wallet failure probability for $n_{stability}$ generations.

Algorithm 2 Evolutionary Wallet Optimization

input : $n, P; n_{pop}, n_{selection}, p_{perturb}, n_{stability}$

$pop \leftarrow n_{pop}$ *random wallets*
$\delta \leftarrow 0$ (Convergence counter)
$best \leftarrow 0$ (Best wallet's probability)
while $\delta < n_{stability}$ **do** (While not stable for long enough)
  if $\max_w\{p^P_{success}(w)|w \in pop\} > best$ **then**
    $best \leftarrow \max_w\{p^P_{success}(w)|w \in pop\}$
    $\delta \leftarrow 0$ (Reset convergence counter)
  else
    $\delta \leftarrow \delta + 1$
  $selected \leftarrow \{\!\{\arg\max_w\{p^P_{success}(w)|w \in pop\}\}\!\}$ (Keep best wallet)
  while $|selected| < n_{pop}$ **do**
    $W \leftarrow$ *choose* $n_{selection}$ *UAR from pop*
    $selected \leftarrow selected + \{\!\{\arg\max_w\{p^P_{success}(w)|w \in W\}\}\!\}$
  $nextGen \leftarrow \{\!\{\}\!\}$
  foreach *wallet* w *in selected* **do**
    $c \xleftarrow{\$} w \cup \{\emptyset\}$ (Choose UAR a combination in w or an empty combination)
    $c' \leftarrow \emptyset$
    foreach $i \in [n+1]$ **do** (With probability $p_{perturb}$ toggle each key)
      with probability $p_{perturb}$
        $c' \leftarrow c' \cup \begin{cases} \{k_i\} & k_i \notin c \\ \emptyset & k_i \in c \end{cases}$
      else
        $c' \leftarrow c' \cup \begin{cases} \{k_i\} & k_i \in c \\ \emptyset & k_i \notin c \end{cases}$
    $w' \leftarrow w \setminus \{c\} \cup \{c'\}$ (Replace perturbed combination)
    $nextGen \leftarrow nextGen + \{w'\}$ (Replace perturbed wallet)
  $pop \leftarrow nextGen$

5 Optimal Wallets

We are now ready to analyze the optimal wallet choices and the resultant probabilities. We begin (§5.1) by studying in detail optimal wallets with small numbers homogeneous keys. Next (§5.2) we look at the effect of the number of keys on wallet failure probability. Finally (§5.3) we turn our attention to optimal wallets for heterogeneous keys.

5.1 Homogeneous Keys

For homogeneous keys we denote state probabilities by P^{safe}, P^{loss}, P^{leak}, and P^{theft}, omitting the key indices.

5.1.1 One Key

For a wallet with a single key, $w^{\mathsf{Single}}(k_1) = k_1$, the probability for success is simply the probability that the key was neither lost, leaked, nor stolen, i.e.,

$$p_{success}(w^{\mathsf{Single}}) = P^{safe} \quad . \tag{4}$$

Figure 1a illustrates the success probability of the wallet for different values of P^{leak} and P^{loss} with a constant $P^{theft} = 0.01$. The isolines show equal failure probability, indicating the lowest failure probability reaches 0.01 when $P^{leak} = P^{loss} = 0$.

5.1.2 Two keys

With two keys several wallets are possible:

AND Require both keys (Figure 1b). For success we require that either both keys are safe, or one is safe and the other is leaked. If either key is lost or stolen then the wallet fails as the owner cannot satisfy it.

$$p_{success}(w_2^{\mathsf{AND}}) = (P^{safe})^2 + 2P^{safe}P^{leak} \quad . \tag{5}$$

OR Require at least one of the keys (Figure 1c). For success we require that either both keys are safe or one is safe and the other is lost. If either key is leaked or stolen then the wallet has failed.

$$p_{success}(w_2^{\mathsf{OR}}) = (P^{safe})^2 + 2P^{safe}P^{loss} \quad . \tag{6}$$

The symmetry between these two wallets is clear, and it is evident that when $P^{leak} < P^{loss}$ the OR wallet is superior, and vice versa. But there is a third option, which simply ignores one of the keys; since the probabilities are the same for both keys it doesn't matter which one:

Single Require that the key is neither lost nor leaked (Figure 1a, Equation 4).

Comparing these probabilities shows that there is a region where using a single key is more secure than either of the other options – when the loss and leak probabilities are similar. To be better than the OR wallet, it should hold that $(P^{safe})^2 + 2P^{safe}P^{loss} < P^{safe}$, and similarly for the AND wallet, i.e.,

$$P^{loss} - P^{leak} < P^{theft} \Rightarrow w_2^{\mathsf{OR}} \prec w^{\text{single}}, \tag{7a}$$

$$P^{leak} - P^{loss} < P^{theft} \Rightarrow w_2^{\mathsf{AND}} \prec w^{\text{single}}, \tag{7b}$$

so if $|P^{loss} - P^{leak}| < P^{theft}$, the optimal 2-key wallet is w^{Single}.

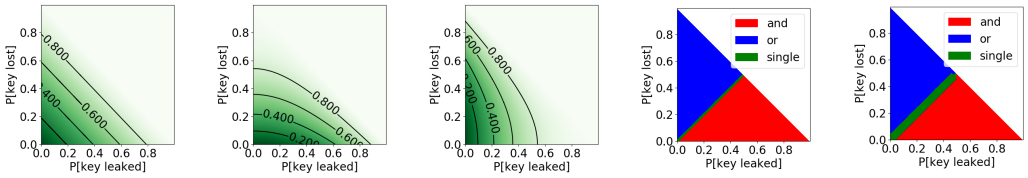
Figures 1d-1e show which wallet dominates each region in the settings space with theft probabilities of 0.01 and 0.04. Note that although we draw the full range (fault probabilities almost up to 1), the practical range is where fault probabilities are small, and the single-key wallet is optimal in a significant portion of this region.

5.1.3 Three keys

With three keys there are 18 possible wallets (§4.1). Figure 2 shows the optimal wallets in a range of settings. In such figures we write i instead of k_i to reduce clutter.

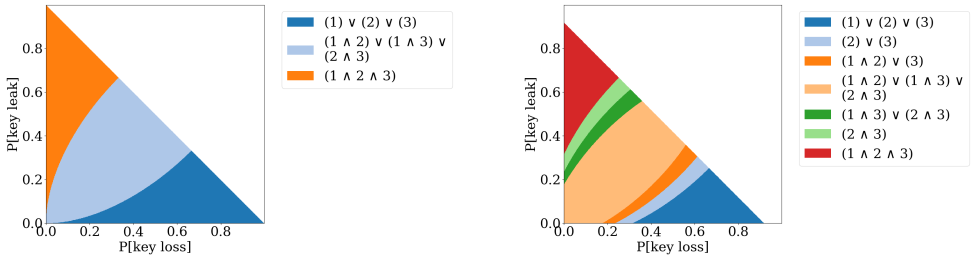
We see that the common wisdom in wallet design [12] is correct in a wide range of settings: The wallet $w^{2/3} = (k_1 \wedge k_2) \vee (k_2 \wedge k_3) \vee (k_3 \wedge k_1)$ is optimal when P^{loss} and P^{leak} are roughly the same. As expected, the OR wallet $(k_1 \vee k_2 \vee k_3)$ is optimal when the loss probability is much higher than the leak probability, and the AND wallet is optimal when a leak is much more likely than a loss.

When there is no theft (Figure 2a), symmetric wallets are optimal across the range of loss and leak probabilities. However, there are regions in the homogeneous settings space where



(a) Failure prob. Single key **(b)** Failure prob. $k_1 \wedge k_2$ **(c)** Failure prob. $k_1 \vee k_2$ **(d)** Best wallet. **(e)** Best wallet; $P^{theft} = 0.04$

Figure 1 Two-key wallets; $P^{theft} = 0.01$ unless noted otherwise.



(a) $P^{theft} = 0.0$ **(b)** $P^{theft} = 0.08$

Figure 2 Three-key optimal wallet for varying theft probability.

asymmetric wallets are optimal, using only a pair of the keys, or all three, e.g., $(k_1 \wedge k_2) \vee (k_3)$. This occurs when theft probability is non-negligible, and becomes pronounced when it is large, as shown for $P^{theft} = 0.8$ in Figure 2b. Nonetheless, the common $w^{2/3}$ wallet remains optimal in the key portion of the space where the fault probabilities are small.

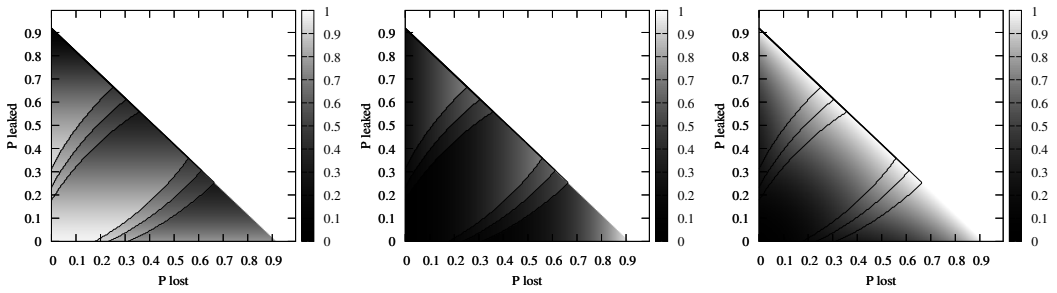
Owner and Adversary Perspectives

To better understand the wallet changes, we explore the probabilities that each of the owner and adversary can satisfy the optimal wallet. A wallet does not fail if the owner has the keys and the adversary does not, and this is the probability we aim to maximize. Figure 3c shows the failure probability of the optimal wallet with $P^{theft} = 0.08$ and varying values of P^{loss} and P^{leak}. The black lines show the transition lines between different optimal wallets (cf. Figure 2). Figures 3a and 3b show the probability that the owner and the adversary (respectively) can satisfy the optimal wallet.

We observe that the wallet success probability is continuous as expected – a switch from one wallet to another occurs at the point where their success probabilities are the same. Figure 3a shows where the owner's decreasing probability leads to a wallet switch, shown as a darkening color switching to light across a boundary. Figure 3b shows the contrary, where the adversary's increasing probability leads to a wallet switch, shown as a lightning color switching to dark across a boundary.

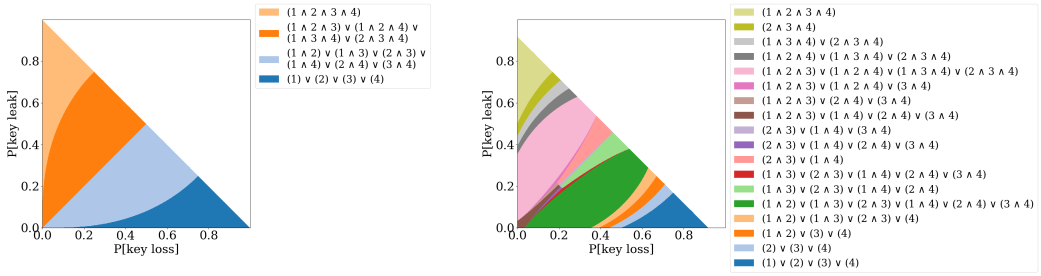
5.1.4 Four Keys

When moving to four keys, it is natural to consider an optimal wallet based on the three-key case, $w^{2/3}$. But extrapolating to four keys we could require either any pair, $w^{2/4}$, or any



(a) Owner access probability **(b)** Adversary access probability **(c)** Wallet failure probability

Figure 3 Wallet failure breakdown. $P^{theft} = 0.08$.



(a) $P^{theft} = 0.0$ **(b)** $P^{theft} = 0.08$

Figure 4 Best four-key wallets without theft.

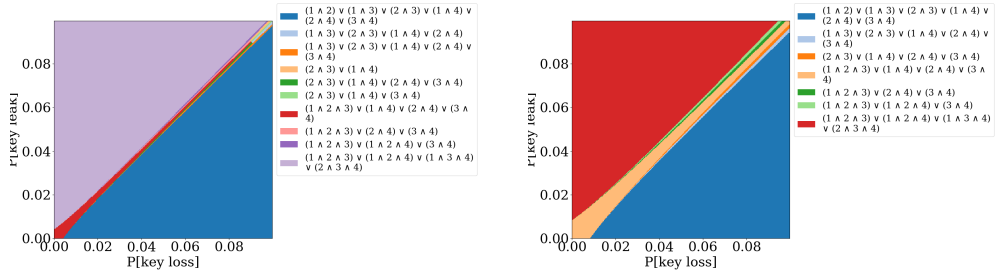
triad, $w^{3/4}$. If there is no theft, Figure 4a shows that indeed these two options dominate a wide range of settings. Specifically, if P^{loss} is larger than P^{leak}, then pairs is the right choice, as less keys are required from the owner. And if P^{leak} is larger than P^{loss} then triads is the right choice, as more keys are required from the attacker. As expected, the OR wallet is optimal when loss is very likely and leak isn't; and the AND wallet is optimal when leak is very likely but loss isn't.

When P^{theft} is positive, as in the two-key case, asymmetric solutions become optimal in a wider range of cases. Figure 4b shows the case for theft probability of 0.08 (taking a large value to illustrate the effect). The two extrapolations of the 3-key case remain dominant in many settings. But now, unless P^{loss} and P^{leak} are significantly different, the optimal wallets are asymmetric.

Figure 5 focuses on the range of smaller fault probabilities, with loss and leak of up to 0.10, and with more realistic theft probabilities of 0.01 and 0.02. We see that in this region, asymmetric wallets become dominant in the most practical region. When both P^{loss} and P^{leak} are roughly similar and below 0.01, the optimal wallet is of the type $(1 \wedge 2 \wedge 3) \vee (1 \wedge 4) \vee (2 \wedge 4) \vee (3 \wedge 4)$ (or any permutation, as the key probabilities are identical).

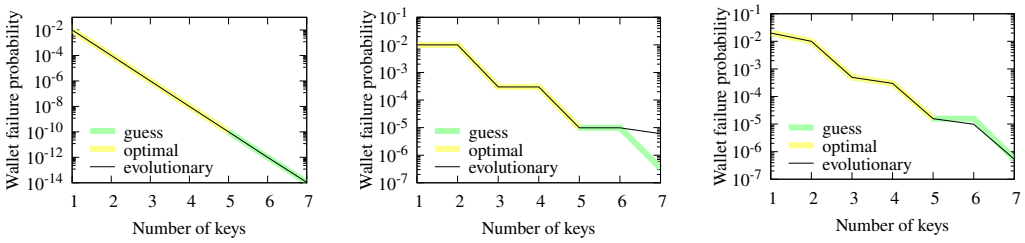
5.2 Number of keys

The number of keys, has major effect on wallet security. While 2-3 keys might be all that is necessary or technically feasible for an individual, much larger numbers could be practical



(a) $P^{theft} = 0.01$ **(b)** $P^{theft} = 0.02$

Figure 5 Best four-key wallets without theft – small probabilities.



(a) $P^{loss} = 0.01$, $P^{leak} = 0.01$ **(b)** $P^{theft} = 0.01$ **(c)** First: $P^{loss} = P^{leak} = 0.01$; Next: $P^{theft} = 0.01$

Figure 6 Wallets with many keys and one fault type.

when storing large amounts, e.g., by financial institutions. In such cases, a group of 6 executives, maybe more, can be assigned to keep 6 different keys.

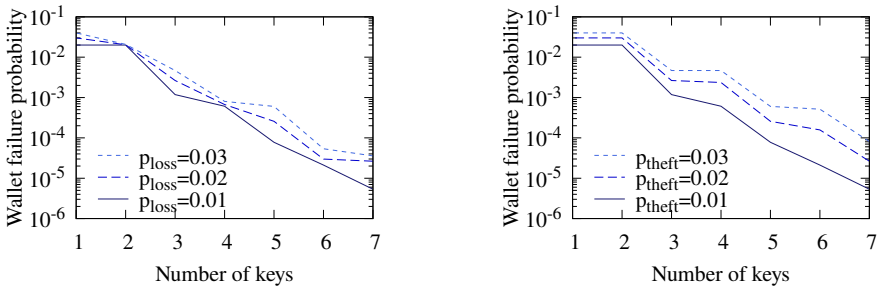
We assume all keys have the same fault probabilities and vary the number of keys from 1 to 7. If the only fault is loss with some probability P^{loss}, the optimal wallet is w_n^{OR}, and its failure probability drops exponentially with the number of keys (Figure 6a). A similar result is achieved with the w_n^{AND} wallet if only leaks are possible.

In a theft-only setting the result is different (Figure 6b). While the overall trend remains exponential, the improvement only emerges on odd numbers of keys. With an odd n number, the best solution is the symmetric $n-1$ optimal solution. The figures show the optimal wallet failure probability, where it can be calculated; with larger numbers of keys they show the failure probability of the best-guess-wallet. The guesses include all symmetric wallets with m keys where $m \leq n$, i.e., all symmetric wallets for smaller key sets.

In all cases the evolutionary algorithm closely approximates the optima when starting from a random population, demonstrating its effectiveness. In the next evaluations we bootstrap with the best symmetric key available, so it can only improve further.

In practice, increasing the number of keys could imply using keys with individual larger fault probabilities. For example, by assigning key keeping responsibilities to more, less trusted, individuals. Figure 6c shows the security when the first wallet can be lost or leaked, but not stolen ($P^{loss} = P^{leak} = 0.01$), and the rest of the keys can only be stolen ($P^{theft} = 0.01$).

Figure 7 shows the approximate optimal wallet failure probability with wallets that suffer only loss and theft with different fault probabilities and different numbers of keys. The exponential improvement with key number results, for example, in an order-of-magnitude



(a) $P^{theft} = 0.01$ **(b)** $P^{loss} = 0.01$

Figure 7 Failure probability with different fault probabilities

	Key 1			Key 2			Key 3			$p_{failure}$	Wallet
	P_1^{loss}	P_1^{leak}	P_1^{theft}	P_2^{loss}	P_2^{leak}	P_2^{theft}	P_3^{loss}	P_3^{leak}	P_3^{theft}		
1	0.100	0.000	0.000	0.100	0.000	0.000	0.100	0.000	0.000	0.0010	$(1) \vee (2) \vee (3)$
2	0.000	0.100	0.000	0.000	0.100	0.000	0.000	0.100	0.000	0.0010	$(1 \wedge 2 \wedge 3)$
3	0.100	0.000	0.000	0.100	0.000	0.000	0.000	0.100	0.000	0.0100	$(1) \vee (2)$
4	0.100	0.000	0.000	0.000	0.100	0.000	0.000	0.100	0.000	0.0100	$(2 \wedge 3)$
5	0.050	0.050	0.000	0.050	0.050	0.000	0.050	0.050	0.000	0.0145	$(1 \wedge 2) \vee (1 \wedge 3) \vee (2 \wedge 3)$
6	0.050	0.050	0.000	0.050	0.050	0.000	0.000	0.100	0.000	0.0122	$(1 \wedge 3) \vee (2 \wedge 3)$
7	0.050	0.050	0.000	0.050	0.050	0.000	0.100	0.000	0.000	0.0122	$(1 \wedge 2) \vee (3)$
8	0.100	0.000	0.000	0.100	0.000	0.000	0.000	0.000	0.100	0.0100	$(1) \vee (2)$
9	0.000	0.000	0.100	0.000	0.000	0.100	0.000	0.000	0.100	0.0280	$(1 \wedge 2) \vee (1 \wedge 3) \vee (2 \wedge 3)$
10	0.000	0.000	0.050	0.000	0.000	0.050	0.000	0.000	0.050	0.0073	$(1 \wedge 2) \vee (1 \wedge 3) \vee (2 \wedge 3)$

Table 3 Heterogeneous Keys

improvement when theft (loss) probability is a constant 0.01, when using 7 keys with loss (theft) probability of 0.03 compared to 3 keys with loss (theft) probability of 0.01.

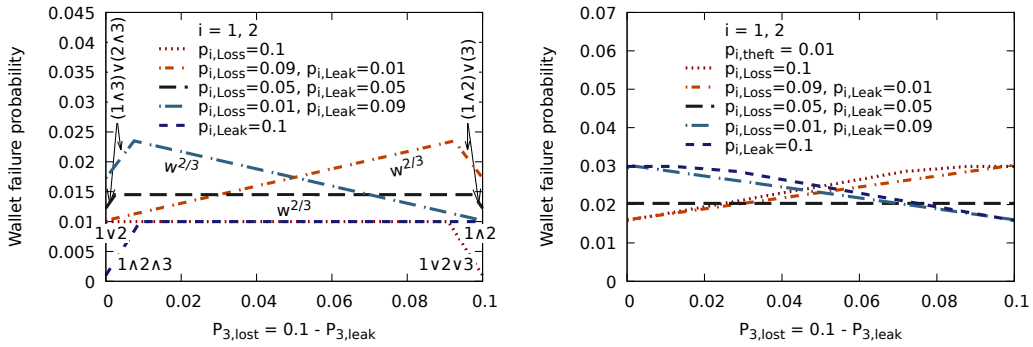
5.3 Key Type Choice

Apart from the number of keys, the owner could choose the types of keys she maintains. For example, she could keep 3 secrets in different safety deposit boxes, or three secrets on hardware devices held by different people. Alternatively, she could keep one key in a safety deposit box, another on a hardware device held by a person, and another memorized.

To compare such alternatives, assume the owner has a budget of 0.1 fault probability per key and can choose three keys such that for each key i: $P_i^{loss} + P_i^{leak} + P_i^{theft} = 0.1$. Table 3 shows the optimal wallet and its failure probability for different heterogeneous key sets. Among these settings, the best wallets are obtained with homogeneous keys that can either be only lost (Line 1) or leaked (Line 2), and theft is naturally the most problematic.

Figure 8 explores the effects of budget distribution with heterogeneous keys. For all key combinations, the sum of fault probabilities is 0.1. In all settings we keep the fault probabilities of keys 1 and 2 constant and vary that of key 3 such that its loss probability grows from 0 to 0.1 and its leak probability drops from 0.1 to 0.

In all cases, if the loss probability of k_3 is very low (and its leak probability is high) it is required in all combinations, as it is unlikely the owner would not have access to it (this is not always visible at this resolution). Similarly, if its leak probability is very low then the optimal wallet can be satisfied only with k_3.



(a) No theft, fault budget of 0.1.

(b) Theft in k_1 and k_2; fault budget of 0.1 for k_3.

Figure 8 Heterogeneous keys.

If the probabilities for the first two keys are pure, i.e., only loss or only leak, it is best to have a third key of the same type. However, if they are not pure ($P^{loss} = 0.09, P^{leak} = 0.01$), it is better to have the third key of a different type (e.g., $P^{loss} = 0.01, P^{leak} = 0.09$).

The optimal wallets change significantly if we introduce just a bit of theft probability ($P_1^{theft} = P_2^{theft} = 0.01$), slightly increasing the fault budget. There is a significant advantage in taking k_3 to be different than the first two, i.e., if they have large loss probability, it is better to take k_3 with small loss probability. With the two keys having $P^{loss} = P^{leak} = 0.05$, all choices of k_3 result in the same wallet failure probability.

6 Conclusion

We present a simple model allowing, for the first time, to analyze the design of cryptocurrency wallets. Our analysis shows that careful design is necessary for constructing secure wallets – adding keys provides in general an exponential improvement, but strongly depends on parity; and choosing whether to add keys of the same type or of a different type depends on the exact key fault probabilities.

Our results raise questions for future work. First, user studies to quantify key fault probabilities are critically missing, both for theoretical analysis and to inform users; the infamous frequency of wallet failures indicates users commonly underestimate the fault probability of their keys. Secondly, the model can be expanded to consider (the undesirable) correlation between key faults. Those occur, for example, if a user keeps two keys at the same location and loses access to it, or if two company employees defect together. Thirdly, our evolutionary algorithm only reaches a limited number of keys, as the key state space became too large to evaluate. Techniques for estimating security with larger spaces would allow to analyze, or at least bound, wallet security with larger numbers of keys.

Nevertheless, the results presented here are of immediate importance to users, who can estimate their keys' fault probabilities, choose keys wisely, and thus improve the security of their funds.

References

1 Myrto Arapinis, Andriana Gkaniatsou, Dimitris Karakostas, and Aggelos Kiayias. A formal treatment of hardware wallets. In *International Conference on Financial Cryptography and Data Security*, pages 426–445. Springer, 2019.

2 Ali Bagherzandi, Stanislaw Jarecki, Nitesh Saxena, and Yanbin Lu. Password-protected secret sharing. In *Proceedings of the 18th ACM conference on Computer and Communications Security*, pages 433–444, 2011.

3 Praveen Baratam. Secure cryptocurrency depository. `https://www.coinvault.tech/wp-content/uploads/2020/10/CoinVault-Secure-Cryptocurrency-Depository.pdf`.

4 Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better, how to make Bitcoin a better currency. In *Financial Cryptography and Data Security*, pages 399–414. Springer, Bonaire, 2012.

5 Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Research perspectives on Bitcoin and second-generation cryptocurrencies. In *Symposium on Security and Privacy*, San Jose, CA, USA, 2015. IEEE.

6 Joachim Breitner and Nadia Heninger. Biased nonce sense: Lattice attacks against weak ecdsa signatures in cryptocurrencies. In *International Conference on Financial Cryptography and Data Security*, pages 3–20. Springer, 2019.

7 Randolph Church. Nunmerical analysis of certain free distributive structures. *Duke Mathematical Journal*, 6(3):732 – 734, 1940. `doi:10.1215/S0012-7094-40-00655-X`.

8 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

9 Richard Dedekind. Über zerlegungen von zahlen durch ihre grössten gemeinsamen theiler. In *Fest-Schrift der Herzoglichen Technischen Hochschule Carolo-Wilhelmina*. Springer, 1897.

10 Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

11 Shayan Eskandari, David Barrera, Elizabeth Stobert, and Jeremy Clark. A first look at the usability of bitcoin key management. In *Workshop on Usable Security and Privacy (USEC)*. Internet Society, 2015.

12 Benjamin Fabian, Tatiana Ermakova, Jonas Krah, Ephan Lando, and Nima Ahrary. Adoption of security and privacy measures in bitcoin–stated and actual behavior. *Available at SSRN 3184130*, 2018.

13 Dinei Florencio and Cormac Herley. Is everything we know about password stealing wrong? *IEEE Security & Privacy*, 10(6):63–69, 2012.

14 Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1179–1194, 2018.

15 Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In *International Conference on Applied Cryptography and Network Security*, pages 156–174. Springer, 2016.

16 Andriana Gkaniatsou, Myrto Arapinis, and Aggelos Kiayias. Low-level attacks in bitcoin wallets. In *International Conference on Information Security*, pages 233–253. Springer, 2017.

17 Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016.

18 Uri Kirstein, Shelly Grossman, Michael Mirkin, James Wilcox, Ittay Eyal, and Mooly Sagiv. Phoenix: A formally verified regenerating vault. *arXiv preprint arXiv:2106.01240*, 2021.

19 Yehuda Lindell and Ariel Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1837–1854, 2018.

20 Malte Möser, Ittay Eyal, and Emin Gün Sirer. Bitcoin covenants. In *International conference on financial cryptography and data security*, pages 126–141. Springer, 2016.

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

22 Marie Vasek, Joseph Bonneau, Ryan Castellucci, Cameron Keith, and Tyler Moore. The bitcoin brain drain: Examining the use and abuse of bitcoin brain wallets. In *International Conference on Financial Cryptography and Data Security*, pages 609–618. Springer, 2016.

23 Morgan Ward. Note on the order of the free distributive lattice, abstract 135. *Bull. Amer. Math. Soc*, 52, 1946.

24 Doug Wiedemann. A computation of the eighth dedekind number. *Order*, 8(1):5–6, 1991.

A Optimal Keys

For two cases we can easily prove what the optimal wallets are.

We often omit the function parameters to reduce clutter. For example, a wallet w and its sub-expressions are all functions of key availability, but rather than writing $w(k_1, k_2, k_3) = e_1(k_1, k_2, k_3) \vee e_2(k_1, k_2, k_3)$ we suffice with $w = e_1 \vee e_2$.

First, if keys can only suffer loss, the optimal wallet is the w_n^{OR} wallet.

▶ **Proposition 1.** *In a setting where all n keys can only suffer loss, i.e., $\forall 1 \leq i \leq n : P_i^{leak} = P_i^{theft} = 0$, for all wallets w, $p_{success}^P(w) \leq p_{success}^P(w_n^{OR})$.*

Proof. Consider a wallet expressed as a DNF comprising m expressions, i.e., $w = e_1 \vee \ldots \vee e_m$, where each e_j is a conjunction of one or more key availabilities.

For any key k_i, consider the wallet w' formed by adding another disjunction of that key, i.e., $w'(k_1, \ldots, k_n) = w(k_1, \ldots, k_n) \vee k_i$.

Set a scenario σ with a positive probability $\Pr[\sigma] > 0$. Key i is either safe or lost. In both cases, it is not available to the adversary, therefore the adversary can satisfy w' iff it can satisfy w. If the owner can satisfy w then it can also satisfy w'. Therefore, if wallet w is successful in σ then wallet w' is also successful in σ. Therefore, the probability that the wallet w' is successful is not smaller than the probability that the wallet w is successful.

We take the wallet w and create a wallet $w_1(k_1, \ldots, k_n) = w(k_1, \ldots k_n) \vee k_1$, and continue creating a series of wallets such that $w_j(k_1, \ldots k_n) = w_{j-1}(k_1, \ldots k_n) \vee k_j$. As we have shown, each wallet is at least as successful as its predecessor and therefore at least as successful as w. The last wallet, $w_n(k_1, \ldots k_n) = w \vee k_1 \vee \cdots \vee k_n$, is equivalent to w_n^{OR}, completing our proof. ◀

Secondly, if keys can only suffer leakage, the optimal wallet is the w_n^{AND} wallet.

▶ **Proposition 2.** *In a setting where all n keys can only suffer leakage, i.e., $\forall 1 \leq i \leq n : P_i^{loss} = P_i^{theft} = 0$, for all wallets w, $p_{success}^P(w) \leq p_{success}^P(w_n^{AND})$.*

The proof approach is similar.

Proof. Consider a wallet expressed as a DNF comprising m expressions, i.e., $w = e_1 \vee \ldots \vee e_m$, where each e_j is a conjunction of one or more key availabilities.

For any key k_i, consider the wallet w' formed by adding k_i to each conjunction, i.e., $w'(k_1, \ldots, k_n) = (e_1 \wedge k_i) \vee \cdots \vee (e_m \wedge k_i)$.

Set a scenario σ with a positive probability $\Pr[\sigma] > 0$. Key i is either safe or leaked. In both cases, the owner can access it, therefore the owner can satisfy w' iff it can satisfy w. If the adversary cannot satisfy w then it cannot satisfy w' either. Therefore, if wallet w is successful in σ then wallet w' is also successful in σ. Therefore, the probability that the wallet w' is successful is not smaller than the probability that the wallet w is successful.

We take the wallet $w = e_1 \vee \ldots e_m$ and create a wallet $w_1 = (e_1 \wedge k_1) \vee \cdots \vee (e_m \wedge k_1)$. and continue creating a series of wallets $w_1, \ldots, w_n$, where $w_j = e_1^j \vee \cdots \vee e_m^j$ and for all $2 \leq j \leq n$ and $1 \leq \ell \leq m$, we take $e_\ell^j = e_\ell^{j-1} \wedge k_j$.

As we have shown, each wallet is at least as successful as its predecessor and therefore at least as successful as w. The last wallet, $w_n = (e_1 \wedge k_1 \wedge \cdots \wedge k_n) \vee \cdots \vee (e_m \wedge k_1 \wedge \cdots \wedge k_n)$ is equivalent to w_n^{AND}, completing our proof. ◀