# Brief Announcement: Make Master Private-Keys Secure by Keeping It Public

Shlomi Dolev[1], Komal Kumari[2], Sharad Mehrotra[3], Baruch Schieber[2], and Shantanu Sharma[2](✉)

[1] Ben-Gurion University of the Negev, Be'er Sheva, Israel
dolev@cs.bgu.ac.il

[2] New Jersey Institute of Technology, Newark, NJ, USA
sbar@njit.edu , shantanu.sharma@njit.edu

[3] University of California at Irvine, Irvine, CA, USA
sharad@ics.uci.edu

**Abstract.** The private key associated with a blockchain is the sole means of linking a cryptocurrency asset to its owner, and any loss or compromise of this key could result in significant consequences. Typically, crypto-wallets generate a private key from a string of words, which users are advised to store in a private record, such as a piece of paper. This method poses several security risks, as the private record holding the secret words can leak the private key. Additionally, private records are vulnerable to being lost or destroyed, leading to the potential loss of assets. Moreover, clients have limited control over the generation of their private key, as the wallet generates it. Our approach empowers clients to securely generate and manage their own private keys, minimizing the risk of key loss. We have developed an open-source technique that allows clients to use memorized secrets to store and retrieve their private keys. Our method employs Bloom filters with hash functions, such as SHA-256, to store and retrieve the private key from the Bloom filter securely.

## 1 Introduction

Private keys of cryptocurrency systems, such as Bitcoin and Ethereum, are the only means of associating the ownership of a client/user with their digital assets. Loss or compromise of the private key can lead to severe consequences, including the permanent loss of funds.[1] To mitigate the risk of losing the private key, clients use cryptographic wallets to store their keys. A crypto wallet can be either a "cold/offline wallet," such as a piece of paper or a flash drive, or a "hot/online wallet," as offered by services like Coinbase and MetaMask. While

[1] https://tinyurl.com/yvxkpk95.

This work was supported by the BGU-NJIT Institute for Future Technologies (seed grant), the Israeli Science Foundation (Grant No. 465/22), the Rita Altura trust chair in computer science, and by the Lynne and William Frankel Center for Computer Science. The work of S. Mehrotra is supported by NSF grants 2420846, 2245372, 2133391, 2008993, and 1952247. The work of S. Sharma is supported by NSF grant 2245374.

T. Masuzawa et al. (Eds.): SSS 2024, LNCS 14931, pp. 338–343, 2025.
https://doi.org/10.1007/978-3-031-74498-3_24

being offline, adversaries cannot access cold wallets; however, they have the risk of being lost/destroyed, preventing the client from accessing their digital assets.

In contrast, existing online wallets provide secure and easy access to digital assets. For every client, these wallets create a private key (a deterministic sequence of 256 bits) derived from a string of secret words, known as the secret recovery phrase. The online wallet generates the secret recovery phrase, consisting of 12, 18, or 24 words, selected from a list of 2048 words [3]. This secret recovery phrase serves as an alternative to memorize the 256-bit private key and must be securely stored by the client. However, in this process, the client neither generates the private key nor selects the secret recovery phrase. To access the wallet, clients need to present the secret recovery phrase to authenticate themselves. The use of these online wallets presents two significant problems:

- *No control to the client.* This major issue arises because the wallet generates both the private key and the secret recovery phrase. For instance, in Coinbase Wallet, clients are provided with an automatically generated 12-word secret recovery phrase, which represents the private key used to access the wallet and perform transactions. Consequently, clients lack complete control over the generation of their private key and secret recovery phrase.
- *Need to remember the secret recovery phrase.* The scheme's security relies on the client's ability to remember the secret recovery phrase presented by the wallet. If clients fail to remember this phrase, they lose complete access to their crypto assets. [5] shows that humans struggle to remember such combinations of words effectively. Humans often store these phrases on a personal computing device, a piece of paper, or in the cloud [4]. However, these options are prone to being misplaced, damaged, or compromised, leading to asset loss. The brain wallet [1], where a user sets a memorable phrase serving as a key, results in choices that can be easily guessed. [6] discovered 884 brain wallets containing 1,806 bitcoins that were compromised due to predictable phrases. Also, brain wallets suffer from the limitations of human memory, which can result in the loss of bitcoins.

This paper tries to address the above-mentioned security concerns of the private key of crypto-wallets in terms of the ***creation and maintenance of the private key*** and asks the following question:

*Is it possible to develop a mechanism that empowers the clients to create their own totally random and never-revealed private keys and store them securely without the risk of being lost?*

**Our contribution.** We develop a technique, entitled R2R (Reminisces to Rescue), that addresses our question. ***The key advantage is that, unlike crypto wallets, our technique leverages the client to create and manage their own private key without the risk of losing it***. R2R uses memorized (possibly very long) secrets, which are different from the private key, and Bloom filters to store/retrieve the client's private key. Humans showcase a great ability to recall ***memorized facts/reminisces/secrets that are unique and***

***known only to the individual***, in contrast to remembering random strings of keywords, such as those generated by existing crypto wallets, as explained above. Examples of memorized and/or owner-retrievable (typically long) secrets could be the first stanza of your favorite song, the fourth paragraph of the third chapter of your favorite book, or a dialogue from your favorite movie or TV show. Unlike the traditional keyword-based secrets, *e.g.*, the first name of your favorite teacher, the name of your first pet, or the last four digits of SSNs, these long-memorized secrets are insusceptible to dictionary attacks. R2R uses Bloom Filter to associate "any" true random private key to private memorized secrets.

R2R enables clients to create their own random bits of private keys, use memorized secrets to store, and (later) extract their private keys. R2R appends a private key after memorized secrets and stores bit-by-bit in a Bloom filter. This results in a pseudorandom sequence of zeros and ones. Such random bits hold no value, unless private memorized secrets of the client are known to adversaries. The client can publish/store replicas of the Bloom filter publicly in newspapers/clouds/local files, avoiding the risk of losing the private key.
**Full version, pseudocode, code in Python, and demo video of R2R technique**: are given in https://tinyurl.com/R2R-Code.

## 2 R2R Technique

**Client and adversarial view.** The client generates its private key, knows security questions provided by R2R, and knows memorized secret answers to the questions. The client executes Insert Algorithm on its private key, resulting in a Bloom filter, which is placed in the public domain, and executes Retrieve Algorithm over the Bloom filter to retrieve the private key. Note that remembering these security questions by the client implies the risk of the client forgetting relevant questions; thus, questions and their order used by the client are also public (as long as the answers are private). An adversary knows the security questions used by client, their order, and the Bloom filter. We call this as *adversarial view*. Based on the adversarial view, an adversary wishes to learn the private key of client. As will become clear soon, the Bloom filter merely appears as a pseudorandom sequence of zeros and ones to adversary, with no meaningful information, unless all private memorized secrets are known to adversary.
**Assumptions:** R2R technique assumes that: (*i*) a client always remembers the memorized secrets, (*ii*) the Bloom filter resides in a public domain, mitigating the risk of it getting lost, (*iii*) the security questions and their order of occurrence are publicly available, and (*iv*) a private key authentication mechanism exists within the online wallet to authenticate client's private key.

### 2.1 Storing Client's Private Key—Insert Algorithm

Our idea is to use public storage and still benefit from the state-of-the-art pseudorandomness implied by the cryptographic hash function, e.g., Secure Hash

Algorithm (SHA). Insert Algorithm encodes the client's private key using memorized secrets and stores the private key in a Bloom filter, which can be published in a public domain. Let $\mathbb{K}$ be a private key of a client. Let $q$ be the number of security questions. Let $a_i$ be the memorized secret answers to the $ith$ question. Let $\mathbb{B}$ be a Bloom filter using hash function $\mathbb{H}$. The client first concatenates all $q$ memorized secret answers: $answer \leftarrow a_1||a_2||\dots||a_q$.[2]

Then, each bit of $\mathbb{K}$ is appended at the end of, one by one, and the resultant sequences are inserted into $\mathbb{B}$ using hash function $\mathbb{H}$ (similar to Bloom filter-based lookup table BFLUT [2]). Finally, $\mathbb{B}$ is placed in the public domain.

**Example of Insert Algorithm.** Suppose, Lisa is a client who wants to store private key 110, selects two questions, and the memorized secrets answers: ($i$) the first stanza of your favorite song, *e.g.*, "You are somebody..." from the song "You Need to Calm Down" by Taylor Swift, and ($ii$) the fourth paragraph of the third chapter of the favorite book, *e.g.*, "Thorndike tracked the behavior..." from the book "Atomic Habits". For simplicity, we are providing a few words from each memorized secret; however, in practice, these memorized secrets will comprise a complete stanza or paragraph. Lisa concatenates the two memorized secret answers as "You...Thorndike...". Then, Lisa creates a Bloom filter $\mathbb{B}$, as: $\mathbb{H}$("You...Thorndike...1"), $\mathbb{H}$("You...Thorndike...11"), $\mathbb{H}$("You...Thorndike...110") by setting one at the corresponding indices. Finally, $\mathbb{B}$ is placed in a public domain. Note that the adversary knows only the two security questions used by Lisa, but not the corresponding secret answers.

### 2.2 Retrieving Client's Private Key—Retrieve Algorithm

The client uses Retrieve Algorithm to retrieve their private key by downloading $\mathbb{B}$ from public domain for performing lookup operations over $\mathbb{B}$. Similar to Insert Algorithm, client first concatenates $q$ memorized secret answers as: $a_1||a_2||\dots||a_q$, resulting in *answer*. To *answer*, the client appends bit zero and then bit one and performs a lookup in $\mathbb{B}$ for the appended sequence—note that bits zero and one can be appended and checked in any order. For each successful lookup ($\mathbb{B}$ outputs as one), the client further appends bit zero, and then bit one and performs lookup for the updated sequence. In case of an unsuccessful lookup ($\mathbb{B}$ outputs zero) the process is terminated for the corresponding sequence. The process continues until the number of appended bits equals $|\mathbb{K}|$, to produce $\mathbb{K}$.

**Example of Retrieve Algorithm.** Consider that Lisa wants to retrieve the private key 110 from $\mathbb{B}$, using the same memorized secret answers, mentioned in the example of Insert Algorithm. Lisa performs lookups: $\mathbb{H}$("You...Thorndike...0"), $\mathbb{H}$("You...Thorndike...1"). Suppose,

[2] An alternative is just to use the output of hash digest, say SHA($answer$), as the private key; however, the result may not be a valid private key for public/private key systems, which is used in the current crypto-wallets. Another alternative to Bloom filter-based solution could use SHA($answer$) to generate a key for AES512, which in turn is used to encrypt and decrypt the signing private key—a private key that is coupled with a paired public key. The Bloom filter solution is more memory efficient when several private keys (say, one for each of the cryptocurrencies) have to be supported. Moreover, the access pattern for retrieving a particular key is less tractable when compared to the access of an entry in a table of encrypted keys.

$\mathbb{H}$("You...Thorndike...1") results in one. Then, Lisa appends zero and one to perform lookup for $\mathbb{H}$("You...Thorndike...10"), $\mathbb{H}$("You...Thorndike...11"). Note, since the lookup of $\mathbb{H}$("You...Thorndike...0") outputs zero, Lisa discontinues the append/lookup process for these sequences. Suppose, $\mathbb{H}$("You...Thorndike...11") results in one. The append/lookup continues until finally Lisa gets the output of $\mathbb{H}$("You...Thorndike...110") as one. Thus, Lisa retrieves $\mathbb{K}$ as 110.

## 2.3 Making False Positives to Zero

Bloom filter lookup comes with false positives with some probability. To avoid false positives, we can append a long sequence of ones or concatenate memorized secret answers at the end of private key. Besides performing insertions as per Insert Algorithm, we append either a sequence of ones or *answer* at the end of $answer||\mathbb{K}$ that results in either $answer||\mathbb{K}||111\ldots$ or $answer||\mathbb{K}||answer$. The sequence of ones is inserted bit-by-bit into $\mathbb{B}$, while *answer* is inserted word-by-word. Note that insertion will increase size of $\mathbb{B}$. During retrieval, once we extract all candidate private keys, say *candidate*, using Retrieve Algorithm. We check all *candidate* appended with ones or *answer* bit by bit, and discard *candidate* whose lookup operation results zero. The process continues until the client is left with a single *candidate* that is the private key of the client.

## 2.4 Security Analysis

The adversarial view constitutes the security questions and their order used by client; however, not the memorized secret answers. Thus, the adversary needs to try **all possible** combinations for stanzas of all the songs and the fourth paragraph from the third chapter of every book. Note that using only one question, such as the first stanza of your favorite song, may make the technique less secure, since an adversary can focus to find this information and learn private key. In contrast, using more than one question, enhances the security of the technique, as adversary needs to learn all the correct answers to learn the private key. In particular, there are over 100M songs on Spotify and over 5M English novels. The adversary needs to try all possible 100M $\times$ 5M $\approx 2^{25}$ combinations to retrieve client's private key. As the client uses $q$ security questions, such that each question has at least a domain of size of 1M, the complexity to learn the memorized secret answers will be at least $(1\text{M})^q$ or $2^{20q}$.[3] Further, the client might consider not disclosing the questions and obfuscating these questions using reminisces in the questions too; e.g., using nicknames or polysemous words. For instance a question "what is best in Israel," could have multiple answers such as city (Haifa), food (Shakshuka), beach (Beit Yanai), actress (Gal Gadot), or TV series (Fauda). The client remembers only one thing that they really like.

[3] Selecting memorized secrets from a large domain is *not* a restriction of R2R. A client can also select memorized secrets from a smaller-sized domain, say 50. In this case, the client needs to select multiple questions. Recall that since questions and their order are available in public, it does not pose a risk of forgetting them. For example, for memorized secrets, each with a domain of size 50, a client may select 20 questions. Here, the adversary needs to try $2^{215}$ combinations, which is computationally infeasible, to learn the memorized secret and then the key.

Suppose, for the client, the best in Israel is Haifa. Based on the keyword "Haifa," the client selects the publicly-known questions $\langle 8, 1, 9, 6, 1 \rangle$ provided by the R2R technique. This method enhances the concealing of the public questions used by clients, hence yielding a practically impossible search for the right answers.

## 3 Conclusion

R2R technique empowers the clients to create their own totally random and never-revealed private keys and store them securely without the risk of being lost. R2R offers security against alphabetically exhaustive searches, preventing an adversary from learning the key. Further, clients do not need to remember the questions used, as they will become public, as well as, long answers (e.g., a book chapter)—the only need is to remember which chapter and then the client can find the chapter. Making questions and Bloom filters public avoids the possibility of losing the private storage, keeping the key secure.

## References

1. Brainwallet. Available at https://en.bitcoin.it/wiki/Brainwallet
2. Dolev, S., et al.: BFLUT bloom filter for private look up tables. In: CSCML (2022)
3. Everything you need to know. https://tinyurl.com/4v682pu
4. What's in the Cloud? Available at https://tinyurl.com/yux3y3yd
5. Ur, B., et al.: "i added '!' at the end to make it secure": Observing password creation in the lab. In: SOUPS, pp. 123–140 (2015)
6. Vasek, M., et al.: The bitcoin brain drain: examining the use and abuse of bitcoin brain wallets. In: FC, pp. 609–618 (2016)