Unforgettable Bitcoin: Recovery through Memorable Secrets

Rarimo protocol, v.0.1

Oleksandr Kurbatov, Lasha Antadze, Yaroslava Chopa

June 15, 2025

Abstract

Secure storage of a private key is a challenge. Seed phrases were introduced in 2013 to enable wallet owners to remember a secret without storing it electronically or writing it down (ideally). Still, very few people can remember even 12 random words. This paper proposes a method for establishing an alternative recovery option that utilizes lower-than-standard entropy secrets (such as passwords, biometrics, and object extractors) in conjunction with Proof-of-Work. We also provide an economic rationale for parameters (e.g., PoW difficulty, secret length, locktime, and cost of the attack) that enable us to achieve the optimal solution from both security and time perspectives.

1 Introduction

Replacing BIP-39 [Bit13] with easier approaches is possible. We have a lot of "user-friendly" key derivation and recovery methods that are based on secrets with lower entropy: passwords, short phrases, biometrics, etc. Of course, these methods aren't comparable to high-entropy keys from a security perspective, so using them in their rough form would be highly insecure. However, we can combine them with additional cryptographic techniques to achieve a trade-off between user experience and security level. The goal of this paper is to investigate such techniques.

We will demonstrate how the user can recover access after forgetting parts of the private key, while maintaining a computational advantage over potential attackers.

1.1 What Can We Break?

Security level is a measure of the complexity of attacking the corresponding cryptographic primitive. We can express it as the number of attempts the attacker needs to break it. If the number of attempts consists of 2^λ, we have a λ-bit security.

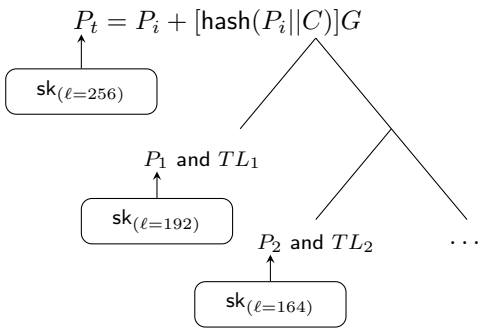
According to FIPS 140-2 [NIS01], the minimum acceptable security strength for cryptographic modules is 112 bits. At the same time, according to Aumasson [KKM19], we have the "broken" category (the primitive can be attacked) with $\lambda \leq 80$ and "wounded" with $80 < \lambda \leq 100$ (can be broken soon). So the minimal acceptable λ should be more than 100 (to make an attack currently impractical), and more than 112 to satisfy current security standards.

Gaëtan Leurent and Thomas Peyrin performed an interesting example of the attack [LP20]. It took $2^{61.2}$ operations to carry out for finding a collision of the SHA-1 hash function with the cost of \$11.000 (they declared ≈\$75.000 but taking into account GPU cost then and preparation cost). So we also want to include the cost of the attack into our calculations (it makes a lot of sense related to the recovery solution we analyze).

2 Bitcoins That Can't Be Lost

Traditionally, Bitcoin relies on the use of 256-bit keys, which provide 128 bits of security (if generated randomly). That's fine for the main access method, but it is impossible to recover funds if this key is lost. With the Taproot [Bit20], we can build a model that reduces the complexity of Bitcoin recovery over time by combining shorter and shorter keys with alternative spending paths, along with appropriate

timelocks (which indicate when alternative secrets can be used). The lower the length of the private key, the lower the difficulty of the discrete log problem.



We understand this approach doesn't help an initial owner in case the key is lost (because everybody can start working on the DL problem and find an appropriate key early). Still, at the same time, BTC from this output can't be lost forever (and we think discrete log competition in the case the primary key is lost is fun).

However, we aim to extend the described approach and provide the fund owner with a significant advantage in solving the discrete log problem based on the set of lower-entropy secrets that can be derived from passwords, biometrics, objects, location, etc.

3 Recovery Approaches

To provide the initial BTC owner with the ability to find an appropriate secret more quickly than others, we can rely on additional knowledge with lower entropy. In this case, the key will include:

1. The part derived from the user's secret
2. The random part needs to be brute-forced

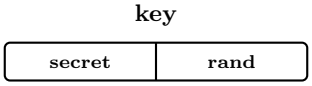


Figure 1: Composite key

Combining this approach with the unlosable bitcoins concept we described earlier, we give the secret owner the advantage to brute-force the needed key much faster (depending on the secret length). While the attacker needs to brute-force the whole key, the owner needs to find the random part (if the initial knowledge wasn't lost).

We can employ two principal schemes: constructing the private key directly from the parts (which we will refer to as a **composite key**) or passing it through the **KDF function**. The first approach is much friendlier because the discrete log problem is easier than brute-forcing KDF's input. Let $s_i, i \in [1, n]$ be a secret components with the lengths ℓ_{s_i} and r a randomness. In this case, the recovery user's key is constructed as:

$$x_r = \sum_{i=1}^{n} (2^{256-\sum_{j\in[1,i]} \ell_j} s_i) + r$$

Having the public key $P_r = [x_r]G$ and secret components, the user can calculate the point R as:

$$R = P_r - [\sum_{i=1}^{n} (2^{256-\sum_{j\in[1,i]} \ell_j} s_i)]G$$

Then the user can solve the DL problem for R to receive the r value and reconstruct the recovery key as described in 3. Let's note that if the user loses one or several secrets from the $s_i, i \in [1, n]$ set, they still are able to reconstruct the secret, but with a more difficult DL problem (depends on the secrets' length)

The second approach is better from a security perspective and more flexible from a time and resource perspective (different KDFs can be used). The recovery key in this case is constructed as:

$$x_r = \mathsf{KDF}(s_1||s_2||...||s_n||r)$$

To recover the key, in this case, the user needs to brute-force the r value in the $[0, 2^{\ell_r} - 1]$ bounds and compare if $[\mathsf{KDF}(s_{i=1}^{n}||r')]G \stackrel{?}{=} P_r$.

4 Recoverable P2TR Address

While the Rarimo team is working on extracting secrets from biometrics and arbitrary objects [Rar24], we provide an example of how the approach might work, based on a password. In the future, we plan to extend our work with the aforementioned secret-retrieving approaches.

4.1 Parameters

Let:

- G: generator of the secp256k1 group of prime order q
- $P_i = [x]G$: internal (primary) public key with secret key $x \in \mathbb{Z}_q$
- p: user password of length $\ell_p \in [16, 28]$ over the 94 printable ASCII characters
- $E = \ell_p \cdot \log_2 94$: raw password entropy (bits)
- $m \in \{0, 1\}^{\ell_m}$: mask, the constant string used for secret padding
- s: salt, a public 128-bit random string (optional, used in KDF).
- c: KDF iteration count
- ℓ_x: target scalar (recovery secret key) length (bits).
- ℓ_k: KDF output length (bits)
- r: the nonce value of the length $\ell_r = l_x - l_m - E$
- T: the absolute timelock height

4.2 Address Generation

4.2.1 Composite Key

1. Construct the recovery key as:

$$k = 2^{\ell_r + \ell_p} \cdot m + 2^{\ell_r} \cdot p + r$$

2. Derive points from the private key parts and aggregate them into one resulting public key:

$$C_m = 2^{\ell_r + \ell_p}[m]G$$
$$C_p = 2^{\ell_r}[p]G$$
$$C_r = [r]G$$
$$C = C_m + C_p + C_r$$

3. Form the Taproot output key

$$P_t = P_i + [\mathsf{hash}(P_i||\mathsf{altRoot})]G.$$

where altRoot includes the script `<<T OP_CHECKLOCKTIMEVERIFY OP_DROP C OP_CHECKSIG>>`, which activates the recovery key only after timelock T.

4.2.2 KDF-based Key

1) Concatenate the mask, password, and nonce into an ℓ-bit string

$$X = (m||p||r) \in \{0,1\}^{\ell_x}.$$

2) Derive the recovery scalar value using KDF:

$$k = \mathsf{KDF}\big(X, s, c, \ell_k\big) \ \in \{0,1\}^{\ell_k}.$$

3) Compute the tweak point

$$C = [k]G.$$

4) Form the Taproot output key

$$P_t = P_i + [\mathsf{hash}(P_i||\mathsf{altRoot})]G.$$

The altRoot is formed in the same manner as in the previous section.

4.3 Recovery Flow

By default, the user is able to access the output if their primary secret isn't lost. If x is lost but p remains, then m and the KDF output k are known, leaving only the nonce r unknown and needs to be found by the user.

4.3.1 Composite Key

In the case of the composite key, the key reconstruction process is easier. Knowing the m and p, the user can subtract appropriate points from the recovery public key and resolve the discrete log problem for the point left:

$$[r]G = C - 2^{\ell_r+\ell_p}[m]G - 2^{\ell_r}[p]G$$

Using Pollard's rho for discrete logarithms requires about N operations, where:

$$N = \sqrt{2^{\ell_r}} = 2^{\frac{\ell_r}{2}}$$

Let $\mathcal{C}$ be the cycle count per scalar multiplication and an a GHz core executes $a \times 10^9$ cycles/s. On an n-core machine, the aggregate rate is

$$\nu = n \times \frac{a \times 10^9}{\mathcal{C}} \text{ ops/s}.$$

Hence, the average recovery time is

$$T = \frac{N}{\nu} \text{ s}.$$

4.3.2 KDF-based Key

If the KDF function was used for the key derivation, the user can't resolve the discrete log only for the r-part. They need to brute-force the randomness on the KDF input, trying to derive the secret corresponding to the public recovery key:

$$[\mathsf{KDF}\big((m||p||r'), s, c, \ell_k\big)]G \stackrel{?}{=} C$$

It takes, on average, 2^{ℓ_r-1} operations; the selection of the KDF function and cycle parameter allows configuring the required brute-force time and complexity (memory and time resistant approaches).

5 Proposed parameters

In this section, we try to fix the parameters of our scheme and calculate the resulting security bounds. We are targeting 256-bit length keys, with the password serving as the primary secret (we indicate $\downarrow$ and $\uparrow$ as the minimal and maximum password length the user uses).

Let the password length be $n_\downarrow = 16$ and $n_\uparrow = 30$, so

$$\begin{aligned} E_\downarrow = n_\downarrow \log_2 94 &\approx 16 \times 6.5546 = 104 \text{ bits} \\ E_\uparrow = n_\uparrow \log_2 94 &\approx 30 \times 6.5546 = 196 \text{ bits} \end{aligned}$$

5.1 Composite Key

Although the user can select any r length, we limit it to 60-72 bits here (from the perspective of the recovery time). In this case, we have:

$$\begin{gathered} \ell_x = 256, \quad \ell_{r\downarrow} = 72, \quad \ell_{r\uparrow} = 60 \\ \ell_{m\downarrow} = \ell_x - E_\downarrow - \ell_{r\downarrow} = 80 \\ \ell_{m\uparrow} = \ell_x - E_\uparrow - \ell_{r\uparrow} = 0 \end{gathered}$$

Hence $\ell_{m\downarrow} = 80$ and $\ell_{m\uparrow} = 0$ (no mask is needed). The discrete-log search complexity for the key owner is

$$\begin{aligned} N_{\mathcal{O}\downarrow} &= 2^{\ell_r/2} = 2^{36} \approx 6.87 \times 10^{10} \text{ group operations} \\ N_{\mathcal{O}\uparrow} &= 2^{\ell_r/2} = 2^{30} \approx 1.07 \times 10^{9} \text{ group operations} \end{aligned}$$

On four $a = 3$ GHz cores with $\mathcal{C} = 547 \times 10^3$ cycles per scalar multiplication, the rate is

$$\nu = n \times \frac{a \times 10^9}{\mathcal{C}} \approx 2.9 \times 10^4 \text{ ops/s},$$

And the average recovery time is

$$\begin{aligned} T_\downarrow &= \frac{N_{\prime\downarrow}}{\nu} \approx 2.37 \times 10^6 \text{ s.} \approx 27 \text{ d.} \\ T_\uparrow &= \frac{N_{\prime\uparrow}}{\nu} \approx 3.69 \times 10^4 \text{ s.} \approx 0.5 \text{ d.} \end{aligned}$$

5.1.1 Attack Cost

When neither the password p (entropy E bits) nor the nonce r (ℓ_r bits) is known, an attacker must exhaustively search the combined space of size

$$N_{\mathcal{A}} = \sqrt{2^{E+\ell_r}} = 2^{\frac{E+\ell_r}{2}}.$$

with the total time

$$T = \frac{2^{\frac{E+\ell_r}{2}}\,\mathcal{C}}{n \times a \times 10^9} \text{ s},$$

The minimum CPU frequency required to achieve $T_{\text{yr}} < 1$ (taking into account the assumption that the recovery key can be activated and should be replaced in 1 year) is

$$\begin{gathered} f_{\min\downarrow} > \frac{2^{\frac{E+\ell_r}{2}}\,\mathcal{C}}{10^9 \times 3.154 \times 10^7} \text{ GHz} \approx 5.35 \times 10^{15} \text{ GHz.} \\ f_{\min\uparrow} > 5.9 \times 10^{27} \text{ GHz} \end{gathered}$$

At an energy cost of $0.71\,\mu$J per scalar multiplication ($\approx 2.6 \times 10^{-14}$ USD), the electricity cost for N trials is:

$$N \times 2.6 \times 10^{-14} = 2^{\frac{E+\ell_r}{2}} \times 2.6 \times 10^{-14} \text{ USD.}$$

For given parameters $\ell_r = 72$ and $E_\downarrow = 104, E_\uparrow = 183$

$$\begin{aligned} 2^{\frac{E+\ell_r}{2}} &= 2^{\frac{176}{2}} \approx 3 \times 10^{26} \downarrow \\ &= 2^{\frac{255}{2}} \approx 8.5 \times 10^{38} \uparrow \end{aligned}$$

Which results in

$$\begin{aligned} 3 \times 10^{26} \times 2.6 \times 10^{-14} &\approx 7.8 \times 10^{12} \text{ USD} \downarrow \\ 8.5 \times 10^{38} \times 2.6 \times 10^{-14} &\approx 2.2 \times 10^{25} \text{ USD} \uparrow \end{aligned}$$

the cost of full brute-force.

5.1.2 Additional Security Considerations

The composite key scheme suffers from several fundamental design flaws that make it less than ideal from a cryptographic perspective:

1. Structural Rigidity: The key construction method, $k = (m << (\ell_r + E)) + (p << \ell_r) + r$ is rigid. It requires a precise a priori estimate of the password's entropy E to determine the ℓ_m and the bit-shift values. If a user chooses a password that is stronger or weaker than anticipated, the scheme does not gracefully adapt. This lack of flexibility compares unfavorably to a KDF-based approach, where the input can be of variable length and is processed into a fixed-length, pseudorandom output.

2. Side-channel attacks: SPA or timing attack could allow an adversary to observe the user's recovery device and make assumptions about the lengths and values of the password and randomness.

3. Structural weakness: the private key k isn't a random integer. Introducing a clear mathematical structure could potentially open up unforeseen attack vectors in the future, especially if novel algebraic attacks on elliptic curves are discovered.

4. Susceptibility to a single, highly structured mathematical problem: the key security in this case is totally based on the ECDLP problem (including the assumption of working with a key length lower than 256 bits). Considering the possibility of hardware acceleration, this scheme is significantly less secure than a KDF-based one.

5.2 KDF-based Key

With the KDF function, we don't need to have 256-bit input data because we are moving from the ECDL problem to the task of collision search. So, for receiving $\lambda = 128$ we can use the 128-bit input secret. We propose referring to a specific KDF function to ensure our calculations are objective; for example, Argon2id [BDK17] is one of the leading KDF functions.

We will target a user recovery time of approximately 30 days on a standard quad-core machine. To achieve this, we select a nonce length of $\ell_r = 26$ and configure Argon2id with parameters that impose a significant computational and memory cost per hash evaluation (e.g., memory cost m=256 MiB, iterations $t = 3$, parallelism $\mathsf{par} = 4$).

The user needs to perform, on average, $2^{\ell_r - 1}$ operations to find the correct nonce. Assuming a quad-core machine can compute approximately 12 of these Argon2id hashes per second, the average recovery time for the user is:

$$T = \frac{2^{\ell_r - 1}}{12} = \frac{2^{25}}{12} \approx 2.8 \times 10^6 \text{sec}$$

An attacker, however, does not know the password p and must therefore search the combined space of the password's entropy E and the nonce's. The total number of operations required by an attacker is, on average:

$$N_{\mathcal{A}} = 2^{E + \ell_r - 1}$$

For even the scenario with the shorter password, we have:

$$N_{\downarrow} = 2^{104+26-1} = 2^{129} \approx 6.8 \times 10^{38} \text{ ops}$$

which is enough. Given that each operation is a memory-hard Argon2id computation, the total work required is astronomically large and computationally infeasible for any known adversary, regardless of hardware specialization (solving EDLP for the final public key is much more efficient in this case).

6 Conclusion and Future Work

The current ideal setup (utilizing a secure KDF function) for $\lambda = 128$ requires a password with 20 random characters (up to 60, taking "usual" people's passwords, which provide not 6.55 but 2,5-3 bits per character). Additional randomness usage (with $\ell_r = 26$) allows for reducing the password to 16 random characters, which is better but still quite hard for users.

The primary goal of our subsequent research is to reduce the length of the user's secret knowledge by utilizing data extracted from the physical world, including biometrics and objects. We believe that replacing the password bits with bits extracted from memorized objects and combining this approach with PoW will lead to a future where people should remember difficult secrets to have their funds recoverable and secure.

References

[NIS01] NIST. *FIPS PUB 140-2: Security Requirements for Cryptographic Modules*. Tech. rep. FIPS PUB 140-2. `https://csrc.nist.gov/publications/detail/fips/140/2/final`. National Institute of Standards and Technology, 2001.

[Bit13] Bitcoin Improvement Proposals. *BIP-39: Mnemonic code for generating deterministic keys*. Online. `https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki`. 2013.

[BDK17] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. "Argon2: The Memory-Hard Function for Password Hashing and Other Applications". In: *IACR Transactions on Symmetric Cryptology* 2017.4 (2017), pp. 43–57. DOI: `10.13154/tosc.v2017.i4.43-57`. URL: `https://tosc.iacr.org/index.php/ToSC/article/view/729`.

[KKM19] Kevin Kelly, Neal Koblitz, and Alfred Menezes. "The Joy of Factoring and the Factoring of Joy". In: *IACR Cryptology ePrint Archive* 2019/1492 (2019). URL: `https://eprint.iacr.org/2019/1492`.

[Bit20] Bitcoin Core Contributors. *BIP-341: Taproot: SegWit version 1 spending rules*. Online. `https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki`. 2020.

[LP20] Gaëtan Leurent and Thomas Peyrin. "SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust". In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2020, pp. 1839–1856. URL: `https://www.usenix.org/conference/usenixsecurity20/presentation/leurent`.

[Rar24] Rarimo. *Bionetta.Ultimate Client-Side ZKML Prover*. GitHub repository. `https://github.com/rarimo/bionetta`. 2024.