

Fuzzy Encryption for Secret Recovery

Esha Ghosh

Daniel Buchner

Jonathan Lee

Melissa Chase

April 2020

1 Introduction

The introduction of Bitcoin led to a renewed interest in decentralized systems backed by users-controlled, user-custodied cryptographic keys. All current implementations of cryptocurrencies, ‘smart contract’ systems, and decentralized identifier protocols require users to maintain secure control over cryptographically random secrets composed of long strings (e.g. 32 byte private keys) that humans are generally unable to remember.

The decentralized systems and applied cryptography communities have generally employed two mechanisms for aiding in the recovery of these unmemorable secrets: 1) randomly selected mnemonic phrases, and 2) Shamir-based secret sharing schemes:

Mnemonic phrase schemes (e.g., BIP 39) generate a set of 12-24 randomly selected words from a corpus that forms a secret seed. While these mnemonic schemes map an unmemorable secret to words, affording some degree of memorability and human error correction (e.g., illegibly written words can be deduced), successful regeneration of the secret requires the user to produce all the words in their exact form and order. These requirements make mnemonic schemes rather unwieldy in practice.

Shamir-based secret sharing schemes use polynomial interpolation as a basis for dividing a secret into N shares, wherein some subset threshold of T shares must be recombined in any order to regenerate the secret. The user can distribute the resulting shares in different ways, for example, storing shares on different devices, hiding paper printouts of shares in different physical locations, or sending shares to a set of collaborators for future retrieval. Typically, this scheme is used for what’s known as ‘social recovery’ models, where you use an app to distribute shares to selected friends or contacts. While this common secret sharing scheme affords the user some flexibility by only requiring them to reproduce a subset of shares, in any order, the shares themselves are long, cryptographically random strings, which makes human recollection of shares impossible.

Both schemes provide various advantages over simply attempting to remember or store long, unmemorable secret strings of random numbers and letters, but the approaches come with almost diametrically opposed trade-offs. With mnemonic phrases, inputs are words, which increases memorability, but users must correctly reproduce all words in the exact order in which they were generated. Secret sharing schemes, on the other hand, only require the user to reproduce a threshold subset of shares, in any order. Unlike mnemonic phrases, however, the shares are long, random strings of letters and numbers, not words or other human-friendly inputs.

The goal of this scheme is to deliver the desirable features of both the aforementioned schemes in one mechanism that allows a user to encrypt a secret with a set of N stringifiable inputs (words, images, etc.), wherein only a T threshold subset of inputs must be recombine in any order to decrypt the secret. The level of security afforded by the scheme is based on a number of factors, including size of the input corpus, entropy of the input selection, and the tolerance of the threshold. The input corpuses, approaches for input selection, and recovery user experiences that produce the best outcomes are still being investigated.

2 Envisioned Technology

We are envisioning a technology that will enable harnessing the entropy in the human-memorizable sets and generate strong cryptographic keys from it, while tolerating a small number of errors. The desirable/required features of our technology are:

1. *No cryptographic secret to memorize:* We will like our scheme to be usable without any need to remember or protect cryptographic secrets. This means, a user is expected to remember/securely protect her pass-phrase only. Any state information generated by the scheme in order to generate the cryptographic key material from the pass-phrase can be stored in any public repository.
2. *Secure recovery:* Only a correct pass-phrase (tolerating a small number of errors) will be able to recover the cryptographic key. The error tolerance threshold is a system parameter. Any pass-phrase that has more than that many errors should fail to recover the cryptographic key or any partial information about it.
3. *Reusability of pass-phrases:* A user should be able to reuse her pass-phrase to generate many cryptographic keys. For example, if a user generates a cryptographic key using a pass-phrase and the key gets compromised, she needs to generate a new cryptographic key. We want her to be able to re-use her pass-phrase to generate a fresh cryptographic key.
4. *Non-iterability:* A pass-phrase will consist of a set of elements, for example “I love to sail forbidden seas, and land on barbarous coasts” is a candidate pass-phrase which consists of a collection of words. We want to have the following property: an adversary who has the public state information will be able to validate its guess for a complete pass-phrase using the state information (by checking if the secret key recovery fails or not), but it should not be able to validate its guess for individual elements in the pass-phrase. For example, an adversary guesses just one word, say, forbidden. It should not be able to learn whether the guessed word belongs to the pass-phrase or not.

3 Our Scheme

We directly use the scheme used in [1] and extend it to satisfy all the features listed above. In particular, the scheme in [1] already satisfies properties 1,2 and 4 listed in Section 2. We extend it to support feature 3. Here we first describe the technical details of our scheme and then discuss how the different parameters contribute build a cryptographically strong system.

3.1 Technical Preliminaries

Each algorithm/party in our scheme, including the potential adversary, has access to the **publicparams**:

- A mapping function H that maps every word in the input domain to a number between 1 and N where N is the size of the domain. Both H, H^{-1} are publicly known.
- A pseudorandom function f (like AES)
- A memory-hard hash function M (like scrypt)
- A finite field F_p of prime order p
- **seed** = $(s_0, s_1, s_2, \dots, s_n) \in F_p^n$ where each s_i is chosen uniformly at random from F_p^n

Input Set Let us denote the human-memorizable set of inputs as \tilde{W} . Let n denote the size of \tilde{W} . Note that the size of the universe for the input set can either be large (superpolynomial in n) or small (polynomial in the size of n). Our proposed scheme builds on the large universe scheme from [1] and is more general than constructions for small universe [1].

In the following scheme, given a set \tilde{W} , it is first transformed to W as follows. Let $\tilde{W} = \{x_1, \dots, x_n\}$. Then, $W = \{H(x_1), \dots, H(x_n)\}$.

Now we recall the Scheme from [1] (Construction 5) and describe the additional steps that we introduce in our scheme.

$\text{GenRecoveryPayload}(\text{publicparams}, W) \rightarrow P$ This algorithm generates a payload that will be used in recovering the W .

$\text{RecoverSet}(\text{publicparams}, W', P) \rightarrow (W/\perp)$ This algorithm recovers W from the payload P as long as $(W - W') \cup (W' - W) \leq t$, t is denoted as the error threshold of the scheme.

3.2 Scheme

Our proposed scheme has the following algorithms. The high level idea of our scheme is to combine the scheme described above with a universal hash function [2] based extractor [1] to first generate a master cryptographic secret key and then use a PRF to derive multiple cryptographic keys from this master secret key.

$\text{SetMasterSecret}(\text{publicparams}, W) \rightarrow (P, \text{counter})$ This algorithm runs $\text{GenRecoveryPayload}(\text{publicparams}, W)$ to generate P . It initializes a counter, sets it to $\text{counter} = 0$ and outputs $(P, \text{counter})$

$\text{GenerateSecret}(\text{publicparams}, W', P, \text{counter}) \rightarrow (\text{sk}, P, \text{counter})$

This algorithms works in the following steps:

1. Run $\text{RecoverSet}(\text{publicparams}, W', P)$ to get W . If RecoverSet fails, fail and abort. Otherwise move to the next step.
2. Sort W . Let $W_{\text{sorted}} = \{a_1, \dots, a_n\}$ denote the sorted set.
3. Compute the following: $e = \prod_{i=1}^n s_i * a_i \bmod p$
4. Compute $ek \leftarrow M(e)$
5. Set $\text{counter} \leftarrow \text{counter} + 1$
6. Finally, a fresh cryptographic key sk is generated as $\text{sk} = f_{ek}(\text{counter})$. The algorithm outputs $(\text{sk}, P, \text{counter})$

$\text{RecoverSecret}(\text{publicparams}, W', P, \text{index}) \rightarrow (\text{sk}/\perp)$

This algorithms works in the following steps:

1. Run steps [1-4] of GenerateSecret .
2. Recover sk as $\text{sk} \leftarrow f_{ek}(\text{index})$.

Security Guarantee Suppose an adversary initially has probability at most 2^k of guessing the input set W from the universe N . Then if the adversary has access to P generated by $\text{GenRecoveryPayload}(\text{publicparams}, W)$, the adversary's chance of guessing W correctly improves only by a little, specifically, to $2^{k'}$ where $k' \leq k - \log \binom{N-n+t}{t}$. (This follows from the guarantees in (Theorem 6.1,[1]).) If the adversary gets to see e.g. a ciphertext encrypted with sk , that may allow him to confirm whether each guess is correct. But even then the adversary will have to compute the memory-hard hash to see whether his guess is correct; an adversary who computes q memory-hard hashes will find the correct sk with probability $\leq (q+1)/2^{k'}$.

Acknowledgements

References

1. Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin. Adam Smith: Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data, EUROCRYPT 2004.
URL: <http://web.cs.ucla.edu/~rafail/PUBLIC/89.pdf>
2. Owen Kaser, Daniel Lemire. Strongly universal string hashing is fast, Computer Journal (2014) 57 (11): 1624-1638
URL: <https://arxiv.org/abs/1202.4961>