Elorm Koto
William Lopez-Cordero
Miranda McClellan
Anthony Rolland

KeyChain

MAS.S62 - 14th May 2018

Background

A rising concern over cryptocurrency wallets is the fact that consumers are required to take full responsibility over authentication credentials; that is, private keys. If the user loses their private key, they cannot recover their funds. Public Key encryption is widely used in security systems; however, the general public's understanding of authentication methods is limited to the use of username and passwords they have experience with. This poses a security risk in itself, in that users will manage private keys with the same degree of carelessness as they manage passwords. Many users might store private keys insecurely on cloud storage, lose the USB or phone the key is stored on, or misplace the paper the key is written on. For this reason key recovery methods are popping up in the crypto world.

Goals

We prototyped a system called KeyChain that alleviates the burden of secure key storage from users. Users can store their encrypted private key on the blockchain by encrypting with it with their fingerprints, a hard to forge data set, and retrieve and decrypt using the same fingerprint. This way, the private key is securely stored on the blockchain and cannot be lost by the user but does not reveal any information about the user to others on the system. Our main goals are as follow:

1. Improve on existing key recovery storage and recovery for cryptocurrency wallets
2. Enable users to recover private keys for accounts without trusted third party
3. Using biometrics to obscure and store private keys securely
4. Allow for secure recovery of private keys from the blockchain using biometric authentication
5. Eliminate the security and privacy concerns for users storing their own private key

Technical Approach

We subdivided the task of creating a easy workflow for secure private key storage on the blockchain into basic sections: obtaining deterministic numbers from user's fingerprint to use to securely store private keys, uploading secured private keys to the blockchain in transactions, and key recovery methods. Lastly, we integrated these steps into an easy to use workflow. We call this set of improvements KeyChain.

Technology

A. Bitcoin testnet3

   Bitcoin test network is used to store transactions made using our new keys. The parameters required to connect to testnet3 are included in the main.go file. Testing the KeyChain system required the use of UTXOs provided by the class.

B. Fuzzy Vault

   Fuzz Vault's original code in python implements a very simple "biometric" authentication system using the fuzzy vault algorithm [1]. Fuzzy Vault provides a few simplifications of the algorithm described by as described by Juels and Sudan in A Fuzzy Vault Scheme [2]. Fuzzy Vault uses a subset of points from a user's fingerprint and elliptic curves to authenticate users We take advantage of the fact that in Fuzzy Vault, the "biometric" data is represented as a list of a small number floats while real fingerprint data is more complex and a large enough set of example data is given.

Implementation

A. Fingerprints

   Our first step was reimplementing the existing Fuzzy Vault in Golang to be compatible with blockchain transaction methods used previously in class. We also were required to rewrite the polynomial regression method in the function, which is significantly different in Go than Python and we found helpful code samples to accomplish this [4]. We use fingerprints as our biometric input. Using our Go version of Fuzzy Vault takes in a fingerprint representation, a list of 10 numbers for simplicity and scale so that it fits within the 520 byte limit of the op_return data. This representation and a chosen private key are used to create a vault which is converted to bytes for the op_return data. The op_return data is then compressed.

B. Transactions

We based our bitcoin transaction methods on the code from *OpReturnTxBuilder* and *GenerateAddress* from pset 3 [3]. From that code, we augmented the methods provided to take in variables like public address, index, sending transaction, receiving transaction, amount, and the compressed vault pieces as input arguments. The methods still produce a transaction for the bitcoin testnet3 and generate a public address, respectively. This allows for additional portability and the ability to use the same methods for multiple transactions without altering the code.

C. Key Recovery

Key recovery is also initiated with the use of a fingerprint. The relevant transactions are retrieved from the blockchain and the op_return data is then combined to obtain the original vault information. Then, the vault is decrypted using the fingerprint data run through Fuzzy Vault and if the fingerprint is valid for the vault, the user can retrieve the private key.

D. Integration

Integration was the most important step for creating a functional system from the fragmented improvements above. Integration is composed of a few basic steps: selecting fingerprint data and private key, creating a vault and compressing it, generating a public address for the blockchain, setting transaction variables and creating a Bitcoin transaction for testnet3 from the variables and the compressed vault data, and including the ability to retrieve the vault data after a transaction has occurred and use the coefficients to decode the private key stored.

The result is KeyChain, a system that joins biometric data and the blockchain for secure private key storage.

The code for this project is available at https://github.com/MirandaMcc/KeyChain-MAS.S62

Potential Optimizations

KeyChain already provides many improvements to existing private key storage solutions. In the future, KeyChain could be enhanced with more features that would allow it to be used efficiently at scale. We have identified key areas where KeyChain could be improved:

1. Scaling op_return data

Currently, the vault created to store the private key is large and grows with the number of points used for validation of fingerprints. Unfortunately, security of the system also grows with the number of points used for validation. Right now, we split the vault data into

chunks and store it on the blockchain across several transactions. Improvements would be to compress this data differently, using a hash of the data to put in the op_return or using a merkle root.

2. User prompts for transaction input

   This will create a more seamless system, but required interaction between two systems (python methods and the bitcoin command line to push transactions) and can be explored in future work.

3. Generation of fingerprints

   This is difficult because fingerprint data is highly sensitive and while we had a fingerprint scanner to use for the project, the process of securely retrieving the data long and required lots of code in C. The process of getting the fingerprint raw data into a useable format for the Fuzzy Vault implementation is also ambiguous because currently this process is proprietary knowledge and not openly shared.

4. Adoption for mobile devices to make transactions “on-the-go”

   This would be very interesting and make bitcoin transactions much more mobile and accessible in daily life like current popular forms of payment like venmo and credit cards. Many phones already include fingerprint scanners, but our ability to access this information is system dependent and proprietary (for example, Apple will not allow access to fingerprints outside of manufacturer’s user authentication purposes).

Resources

1. https://github.com/jwoogerd/fuzzy_vault
2. http://people.csail.mit.edu/madhu/papers/2002/ari-journ.pdf
3. https://rosettacode.org/wiki/Polynomial_regression
4. https://github.com/mit-dci/mas.s62/tree/master/pset03