



COMP6451 - Cryptocurrency and Distributed Ledger Technologies

Lecture - Key Management & Wallets

Ron van der Meyden

UNSW School of Computer Science and Engineering

Summary

Topics:

- Key Management
- Wallets
 - Hierarchical wallets
- Robust key storage:
 - Secret Sharing
 - Multisig/
 - Threshold cryptography

Bitcoin's key management headache

Recall:

- Bitcoin has no accounts
- For security and privacy purposes, value is controlled by a private key
- For better privacy, *for each new amount of value received*, you should create a *new* private key

Managing all these keys is hard for users.

Wallets aim to provide a more practical user interface to help users with the task.

(“*Key-ring*” would have been a better name, since wallets don’t store money directly!
They store keys that unlock access to money!)

Cold Storage

The split between public and private keys is helpful for security:

High Security Cold Storage

Private Keys w.
large amounts,
Transaction Signing



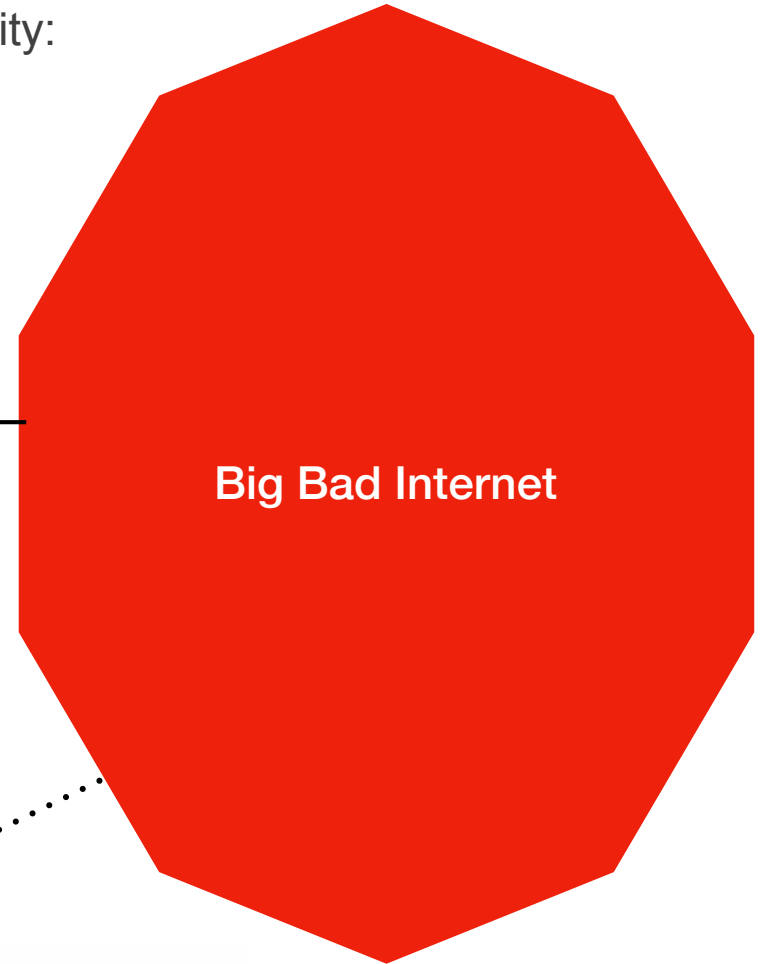
Airgap

Less Secure Server

Public Keys
“send money
here”



Big Bad Internet



Mobile Device “Hot wallet”

Private Keys w.
small amounts



Paper Wallets

- Key Generation performed on the user's PC
(while *disconnected from any network*, though this still gives limited security in case of a compromised machine.)
- Some web-pages support this (trust required!)
- Print Public Address/Private Key in Base58Check
- QR code for easy use

Public Address



Private Key

Example from <https://coinsutra.com/make-bitcoin-paper-wallet-spend-bitcoins/>

Mnemonic Code Words

Base58Check encoded private keys are strings such as

```
5Kb8kLf9zgWQnogidDA76MzPL6TsZZY36hWXMssSzNydYXYB9KF
```

This is shorter than a hex-encoded version, but still far from acceptably user friendly:

- hard to remember,
- painful when the user has to type it in!

Mnemonic code words encode bit sequences as a more memorable sequence of words

BIP0039 proposes a way to do this (used by some wallets) :

- Divide the check-summed key into short segments (e.g. 11 bits each)
- Use those segments to index into a dictionary of words (e.g. of 2048 words)
- Replace the segments by the corresponding words

Example:

apple fox explain mother print hero alpha large create gold water zero west door high

Brain Wallets

“Brain Wallets” also use a sequence of words to represent a key, but rely on the user to come up with the sequence of words.

The key is derived as

hash(word-sequence)

- This makes the sequence easier to remember.
- It also makes it ***MUCH less secure***.

Critical Principle: Keys should be generated *randomly*, using a *good source of randomness*.

Software Wallets

Software wallets store private keys on your device, so are only as secure as that device.

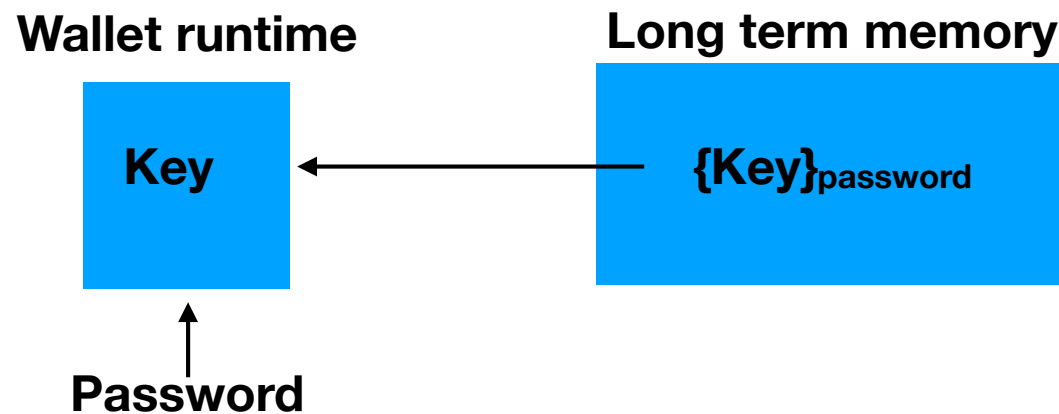
You are trusting the hardware, operating system, third party libraries (& possibly browser) to prevent keys leaking to malicious processes on your machine.

Prominent Bitcoin Software wallets:

- The Bitcoin Core reference client comes with a software wallet
- Electrum
- Jaxx
- Atomic

Key Passwords

Encrypting the private key using a password for storage gives some protection.



- Protection is only as strong as the password (& keyboard snooping protection)
- Wallet app should be written to erase key material once no longer needed

Hardware Wallets

Hardware Wallets are special purpose hardware devices

- Designed to offer high security generation and storage of private keys
- Does not allow private key to escape the device
- Signatures performed *on* the device
- Only public keys/addresses/signatures allowed to escape device
- User interaction with device may be partially through app on less secure desktop/mobile
- Simple (LED) display for messages to user
(e.g. “click to approve transfer of 1BTC to address XXX”)
- Design and features deliberately kept very simple, so easier to ensure security
- May use special purpose hardware chips designed for high security applications

Hardware Wallet Examples



Trezor

Nano Ledger



Nondeterministic/deterministic Wallets

The early wallets were *nondeterministic* wallets, that generate each new key using the random source

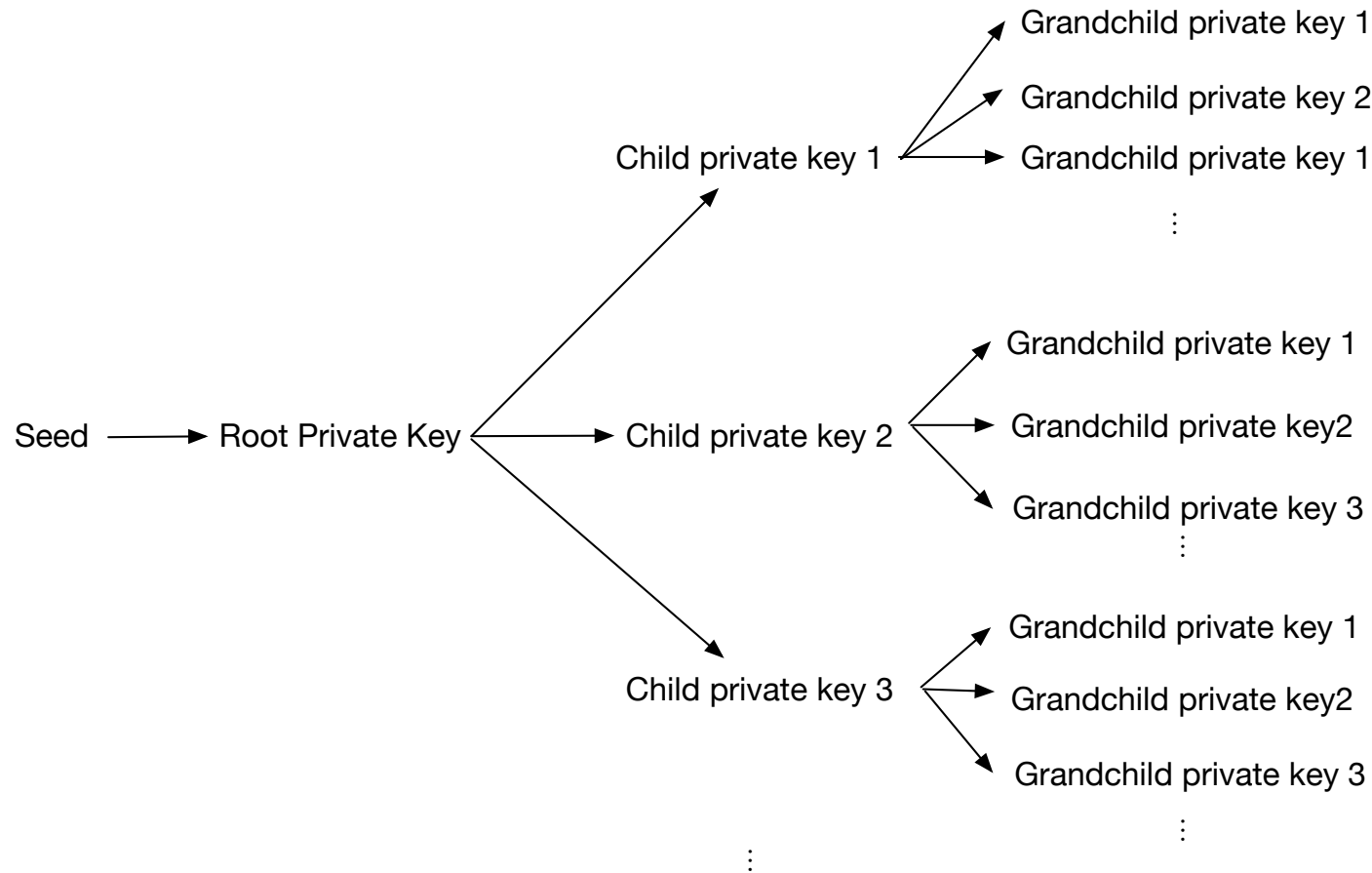
This offers the best security, but comes with some inconvenience:

- More key material to be backed up
- More frequent backups needed
- Harder to keep track of backup status
- Frequent need to transfer key material from cold to hot storage

Hierarchical Wallets

Deterministic or seeded wallets generate batches of keys from a shared random *seed*

Hierarchical Wallets are deterministic wallets that do this in the following structure, useful for hierarchically structured organisations.



Hierarchical Wallet Math

Recall, that for ECDSA,

public key = $g^{\text{private key}}$ where g is a public parameter.

so public keys can always be derived from private keys.

Root private key derivation:

seed — SHA512 —> (root private key (256 bits), root chain code (256 bits))

Child private key derivation:

parent public key || parent chain code || index i — SHA512 —> (x_i , child i chain code)

child i private key = parent private key + x_i

Public Derivation of Public Key

These definitions have a nice property with respect to calculation of child public keys:

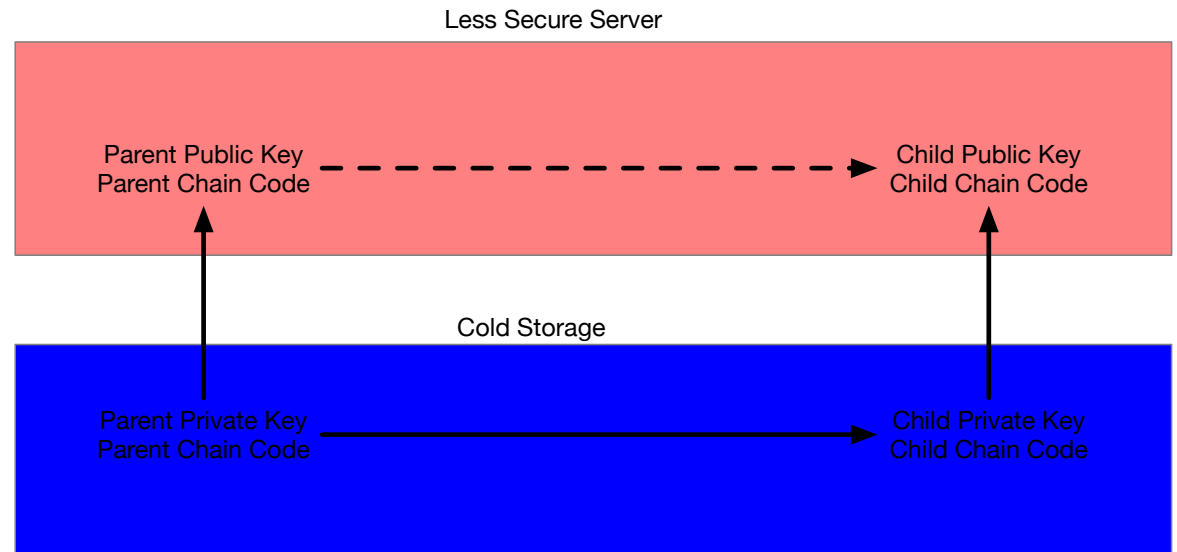
child i public key

$$= g^{\{\text{child i private key}\}}$$

$$= g^{\{\text{parent private key} + x_i\}}$$

$$= g^{\{\text{parent private key}\}} \cdot g^{\{x_i\}}$$

$$= \text{parent public key} \cdot g^{\{x_i\}}$$



This enables child public keys to be derived without needing to extract the private key from cold storage!

Hardened Private Key derivation

A risk of hierarchical key derivation:

if a child private key and the corresponding chain code are ever lost to an attacker, then the attacker can efficiently derive *all* private keys in the subtree from that node.

- child private key , parent chain code $\rightarrow$ parent private key

Hardened child key derivation :

parent private key || parent chain code || index i $\xrightarrow{\text{SHA512}}$ (x_i , child i chain code)

child i private key = parent private key + x_i

Secret Sharing

Long term storage in a single location of private keys that lock money is risky:

- compromise of the location => the attacker can steal the money
- failure/destruction of the storage medium => the money is lost forever

Secret sharing provides a way to manage this risk.

A simple secret sharing scheme:

Split the key into n parts and distribute the parts to n different storage locations.

Recombining the n shares reconstructs the key.

Now an attacker would have to compromise n locations to get the key.

k/n-Secret Sharing

Problem with the simple secret sharing scheme:

Compromise of one location does significantly decrease the cost of a brute force attack, by halving the number of bits to be guessed.

Failure of any *one* of the locations => loss of the key.

This increases the risk n -fold!

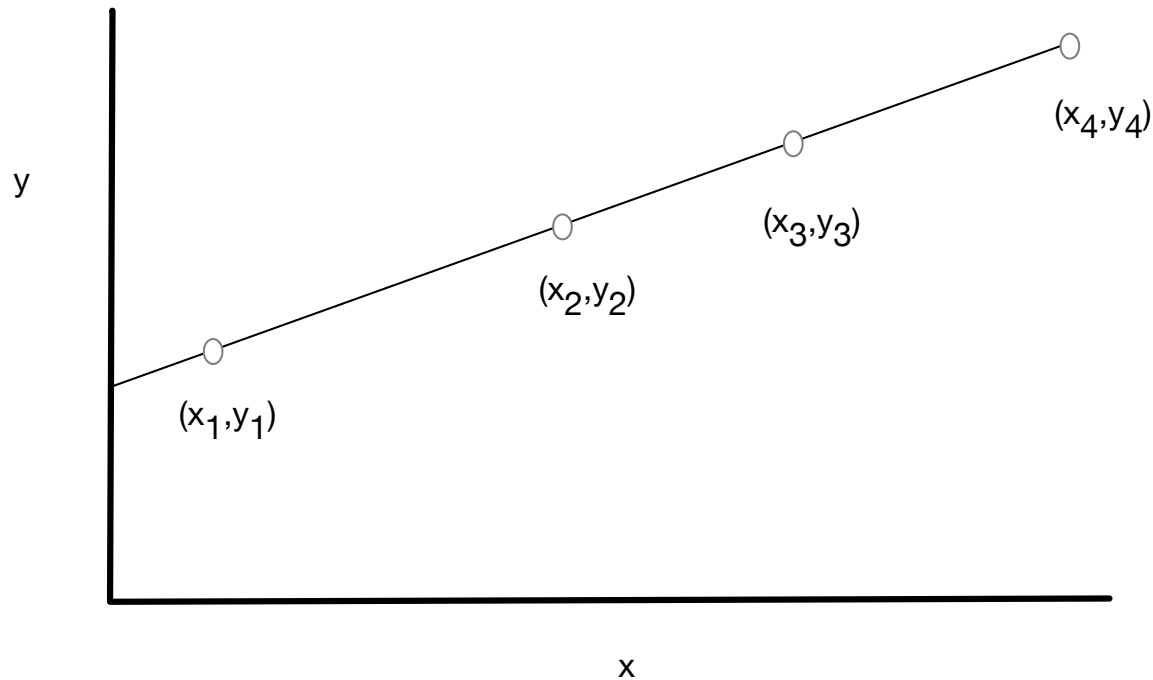
A k/n secret sharing scheme is a way to

- split a key into n shares, such that
- any k of the shares suffice to reconstruct the key.
- fewer than k shares *do not suffice* to reconstruct the key
(or, even better: *give no information* about the key)

2/n Secret Sharing

A 2/n secret sharing scheme can be constructed as follows

- We view the key as a pair (a,b) defining a straight line $y = ax + b$
- The shares are n points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ on this line



k/n Secret Sharing

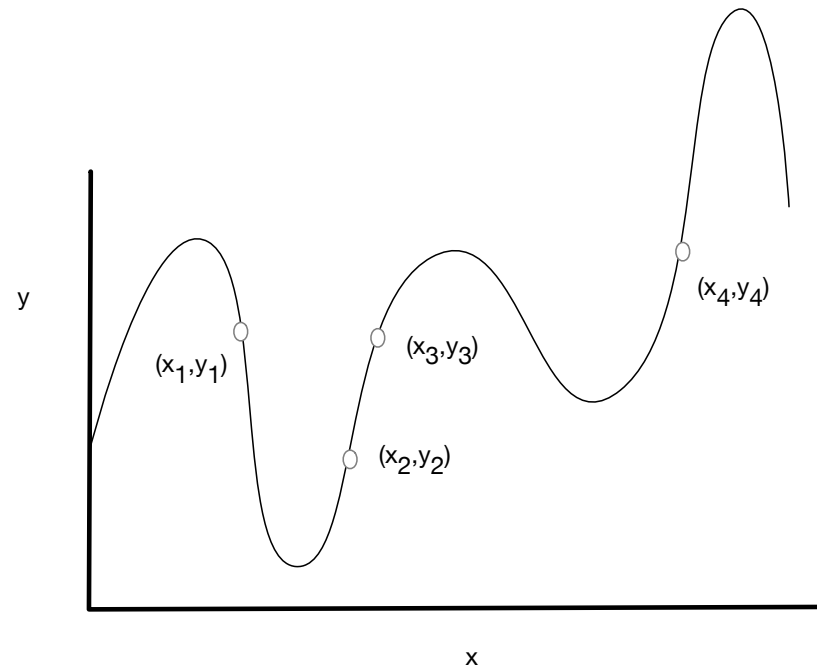
This idea generalises to a k/n secret sharing scheme:

- We view the secret key as numbers $(a_{k-1}, a_{k-2}, \dots, a_0)$, representing a degree $k-1$ polynomial:

$$y = a_{k-1} x^{k-1} + a_{k-2} x^{k-2} + \dots + a_1 x + a_0$$

- The shares are n points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ on this polynomial

Mathematical fact: any k points on a degree $k-1$ polynomial uniquely determine the coefficients $a_{k-1}, a_{k-2}, \dots, a_0$.

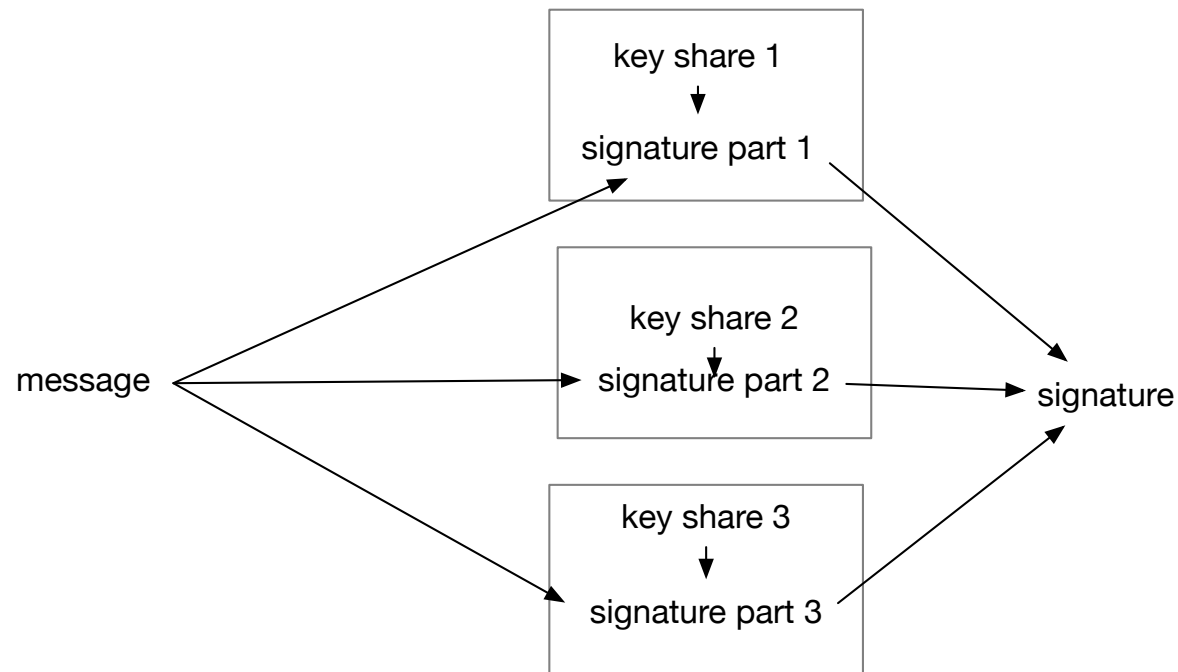


Threshold Cryptography

Even if we use secret sharing, we need to bring the private key shares together at one location to compute a signature.

If that location is compromised by an attacker, then the private key will be compromised.

Threshold cryptography gives a way to encrypt/sign in a decentralised way, without needing to bring the shares together.



Multi-sig

Bitcoin script can be used to implement a *multi-signature* capability that is similar to decentralised k/n threshold signatures.

Let $A_1, \dots, A_n$ be n Bitcoin addresses, corresponding to private keys $K_1, \dots, K_n$

We use a transaction output with an unlocking script that expresses the following:

unlock when provided with k distinct pairs $(P_i, \text{<sig>}_i)$ where

- each P is a public key with address $\text{hash}(P)$ in the set $\{A_1, \dots, A_n\}$
- each <sig>_i is a signature of $\text{hash}(\text{spending transaction})$ that verifies using P_i .

The keys $K_1, \dots, K_n$ act like shares, but they don't need to be located in a common location and distributed — they can be generated independently at the distributed nodes.

Common application:

Both the company secretary and the CEO need to sign to release these funds .