

Deterministic wallets

Gregory Maxwell et. al.

Presented by Iddo Bentov

Bitcoin Israel Conference

February 27, 2014

Main ideas

One-sentence summary

- Instead of generating random-independent Bitcoin addresses in your wallet file, use just one secret value as a seed that generates a pseudorandom (deterministic) sequence of values, and derive a Bitcoin address from each value in this sequence.
- Example: $\text{privkey}_1 = \text{hash}(\text{seed}||1)$, $\text{privkey}_2 = \text{hash}(\text{seed}||2)$, ...

Twist

- By using a key homomorphism feature of discrete-log based cryptosystems (in particular ECDSA), we can generate the public-key values of this deterministic sequence without knowing their corresponding private-keys (and with no need to access the master seed or any other highly sensitive data).

The reasons why multiple addresses are necessary

Why does a Bitcoin user need to maintain multiple addresses?

- Receiving Bitcoin payments: it is needed to allocate a unique address to each customer in order to tell who sent which payment (the customer's transaction cannot readily be signed with an identity, because it may consist of multiple inputs).
- Anonymity: even if you personally don't mind being identified, the receiver of the payment may prefer not to have her customers associated with other organizations.
- Don't put all your eggs in one basket: if one private-key is compromised (e.g. due to a sidechannel attack), the coins that are controlled by your other private-keys are unaffected.
- Security: if you re-use addresses, your coins are protected with 128 bits of security, rather than 160 bits (and unstructured symmetric crypto is less prone to attacks), except while your transaction is broadcast but not yet buried under enough PoW.
- Privacy: better not reveal how many coins you possess.

Random-independent vs deterministic wallets under common use




Random-independent wallets:

- Constant worries: the user needs to keep creating additional wallet backups, as new receiving / change addresses are being created.
- The simple task of generating a new receiving address requires you to access (decrypt) your most sensitive data.
- Need a good source of randomness on lite devices (Android phones avoid using multiple addresses...)

Deterministic wallets:



- Foolproof: backup your grandma's master seed just once, and you can be sure that she can retrieve all her future private-keys.
- Access your secret data only when it's absolutely necessary, i.e. when you spend your coins by signing with your private-keys.
- No randomness required.


Example with 7208 bitcoins lost due to backups mixup (+client bug)

→   <https://bitcointalk.org/index.php?topic=11104.0> 

Pages: [1] 2 All « previous topic next topic »
 reply | watch | notify | mark unread | print

Author Topic: **Lost Savings Wallet Addresses?!** (Read 9086 times)



Sad Puppy
 Newbie
 Activity: 8
 Ignore


 **Lost Savings Wallet Addresses?!** [quote](#) #1
 June 01, 2011, 05:10:56 PM

I think I just managed to lose a large number of BTC. Here's what happened:

1. I had a wallet with all my BTC. I quit Bitcoin (version 0.3.21) and renamed the entire Bitcoin directory Bitcoin-checking.
2. I re-opened Bitcoin, which created a new Bitcoin directory and downloaded all the blocks again.
3. I copied the address shown, quit Bitcoin, renamed this directory Bitcoin-savings, encrypted it as Bitcoin-savings-encrypted, and saved it in multiple remote locations.
4. I renamed Bitcoin-checking to Bitcoin, then restarted the Bitcoin application.
5. I sent 0.02 BTC to the address from step 3.
6. I quit Bitcoin and renamed the Bitcoin directory to Bitcoin-checking.
7. I unencrypted a copy of Bitcoin-savings-encrypted, renamed the directory to Bitcoin, and restarted the Bitcoin application.
8. My 0.02 BTC showed up in this savings wallet.
9. I copied another address, quit Bitcoin again, renamed the directory as Bitcoin-savings, swapped in Bitcoin-checking and sent lots of BTC to this new savings address.
10. I never updated the Bitcoin-savings-encrypted file after step 3, because I thought the wallet automatically contained 100 pre-generated addresses to start.
11. I securely deleted my unencrypted Bitcoin-savings directory with multiple passes.
12. Later I unencrypted a copy of the Bitcoin-savings-encrypted directory, renamed it Bitcoin, opened the Bitcoin app, and only my original 0.02 BTC are shown even after all the new blocks are downloaded.

So it looks like I lost all the BTC I transferred to my savings wallet! I downloaded bitcointools from here:
<https://github.com/navinandresen/bitcointools>

Sad Puppy
 Newbie
 Activity: 8
 Ignore


 **Re: Lost Savings Wallet Addresses?!** [quote](#) #7
 June 01, 2011, 06:01:38 PM


Quote from: fornit on June 01, 2011, 05:49:49 PM
 Quote from: Sad Puppy on June 01, 2011, 05:34:18 PM
 I would edit the Wiki myself, but I'm busy contemplating jumping out a window in despair.

was it that much?

Yes, unfortunately. I hate to admit it, but I lost 7208 BTC. At least for me, that's a lot. My savings just got destroyed. Uggggghhhh. I seriously can't believe it. I'm totally screwed.

[Report to moderator](#)

theymos
 Administrator
 Hero Member
 Activity: 1484

 **Re: Lost Savings Wallet Addresses?!** [quote](#) #8
 June 01, 2011, 06:07:08 PM

There is a bug where the creation of the first-ever address (with label "Your Address") does not trigger filling of the keypool. The keypool is only filled once the *next* address is created, and then it takes tens of minutes to fill completely.

[Report to moderator](#)

Drawbacks of deterministic wallets?

The (non-)problems of deterministic wallets

- Single point of failure: if your master secret (seed) is compromised, all your coins will be stolen:
 - To regular users, this risk isn't much different from stealing their unencrypted wallet.dat file with all the private-keys.
 - We can do better: hierarchical sub-wallets, delegated sub-wallets via key homomorphism (+multisig secret sharing, deniable encryption that reveals a decoy low-value sub-wallet).
- Sound crypto? Is there a related-keys attack?
 - **Short answer: no.** An existential forgery attack implies a distinguisher between pseudorandom and random bits.
 - For example, consider two 5-megabyte files f_1 and f_2 :
 - $seed = (128 \text{ purely random bits})$
 - $f_1 = (\text{SHA256}(seed||1), \text{SHA256}(seed||2), \text{SHA256}(seed||3), \dots)$
 - $f_2 = (5 \text{ megabytes of purely random bits})$

If you think that it is infeasible to distinguish between f_1 and f_2 , then you also think that deterministic wallets are secure.

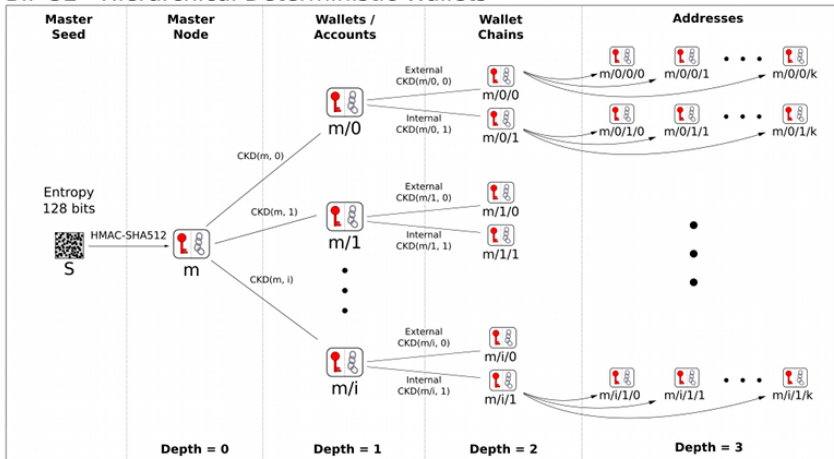
Public/private key homomorphism

Discrete log cryptosystems with private/public key homomorphism

- ECDSA: $G = (x, y)$, $n \cdot G \triangleq \overbrace{G + G + \dots + G}^{n \text{ times}}$
 - $pubkey_1 = privkey_1 \cdot G$
 - $n_i \leftarrow \text{hash}(\text{derivations_nonce} || i), i = 2, 3, 4, \dots$
 - $pubkey_2 \leftarrow pubkey_1 + n_2 \cdot G$
 - $\Rightarrow (privkey_1 + n_2) \cdot G = privkey_1 \cdot G + n_2 \cdot G = pubkey_1 + n_2 \cdot G$
 - $\Rightarrow privkey_2 = privkey_1 + n_2$ **note: $privkey_2 \in \{1, 2, \dots, \text{order}(G)\}$**
 - $\Rightarrow privkey_i = privkey_1 + n_i, pubkey_i = pubkey_1 + n_i \cdot G$
- ✓ DSA: $pubkey_1 = g^{n_1} \bmod p, g^{n_1+n_2} = g^{n_1}g^{n_2} = pubkey_1 \cdot g^{n_2}$
- ✗ RSA: $pubkey_1 = (pq, e), privkey_1 = d$
- ✗ Lattices, typically: $pubkey_1 = A \in \mathbb{F}_p^{n \times k}, privkey_1 = S \in \mathbb{F}_p^{n \times k'}$

The default hierarchical structure of a deterministic wallet

BIP 32 - Hierarchical Deterministic Wallets



$$\text{Child Key Derivation Function} \sim \text{CKD}(x,n) = \text{HMAC-SHA512}(x_{\text{Chain}}, x_{\text{PubKey}} \parallel n)$$

Homomorphic derivations

Use case of homomorphic key delegation

As a merchant, you probably run a server computer so that each customer connects to this server and receives a Bitcoin payment address. The server can store only a master public key (and chaincode), and derive the payment addresses for the customers.
⇒ If the server is breached by hackers, they cannot steal any coins.

The danger with homomorphic key derivations

- $pubkey_{i+1} \leftarrow pubkey_i + n_i \cdot G$ (n_i is the chaincode)
- Suppose that $privkey_{i+1}$ is compromised, and that the chaincodes are public or known to the attacker somehow.
- Solve for x : $x + n_i = privkey_{i+1} \pmod{order(G)}$
- ⇒ if a single private key is compromised, all the other keys in the wallet become compromised too.
- This isn't the case with random-independent wallet keys...

Non-homomorphic derivations

How to mitigate the risks of homomorphic derivations

- Use a simple non-homomorphic derivation:
 $privkey_{i+1} \leftarrow \text{hash}(\text{chaincode} || i + 1)$
- This way, if $privkey_{i+1}$ is compromised, only the sub-wallet that is rooted at $privkey_{i+1}$ becomes known to the attacker.
- In the example with the server computer, the merchant may wish to use non-homomorphic derivation for the root of the server's sub-wallet. Hence, if the server is breached, an attacker who obtained a private key (from elsewhere!) can only steal the coins in the server's sub-wallet.
- The default hierarchical structure uses non-homomorphic derivations for "accounts" at depth 1, and homomorphic derivations in each account.

Brain wallets and paper wallets

Brain wallet 2.0 ? Paper wallet 2.0 ?

- Brain wallet 2.0: Instead of a random master seed, the user can derive the root node at depth 0 via a passphrase.
- This way, the user can re-gain access to all of her coins by scanning the tree structure of her hierarchical wallet (trivial to do with the default structure).
- Example with “security through obscurity”:
 $seed = \text{SHA3}(\text{SHA256}(\text{SHA256}(\text{SHA256}(\text{passphrase}))))$.
- We can do better, see: self-descriptive strengthened keying, or halting password puzzles.
- If you send bitcoins to a private key that is derived as $\text{SHA256}(\text{dictionary_word})$, the coins will be stolen instantly...
- For paper wallet 2.0, the user should print a seed of at least 128 random bits, and at most 512 random bits.

Multi-signature scripts w.r.t. homomorphic derivations

The hierarchal deterministic wallet standard is a standard for generating cryptographic keys, rather than a standard for generating Bitcoin scripts.

BIP16+BIP32 ?

- For extra security, suppose that you wish to be able to redeem your coins via a script of the following kind:
 $OP_CHECKSIG(pubkey_1, \cdot) \text{ AND } OP_CHECKSIG(pubkey_2, \cdot)$
- For example, $privkey_1$ may reside on your laptop and $privkey_2$ may reside on your smartphone.
- The server that derives the (P2SH) payment addresses for your customers can store two public root nodes (i.e. each node consists of a public key and a chaincode), and assemble a script of the needed format for each customer.

Future plans

BIP32 \neq BIP32.5

- Vanilla ECDSA makes use of a random value k for each signature that you create.
- It is better to have deterministic signatures (Android bug...), both because of insufficient randomness on lite devices, and to test implementations across systems.
- Basically $k = \text{hash}(\text{privkey}||\text{message})$
- The Bitcoin developers plan to switch to deterministic signatures (this is unrelated to deterministic wallets).

Stealth addresses

- Allows having a single public address, where each user can anonymously derive a payment address from this single address.
- This is done via Diffie-Hellman key exchange...

Thank you.