Reusable Addresses

Jumping around in Key Space with Non-Interactive Diffie-Hellman Key Exchange

Kaspar Etter, November 2018

SHIFT CRYPTOSECURITY

Motivation

In Bitcoin, keys should never be reused because:

- **Privacy**: All transactions are publicly visible.
- **Security**: Until an output is spent, only the hash of the public key is stored on the blockchain. If ECDSA breaks, the output is still protected by the hash function (also more bits of security).

Problem: We often need to embed addresses in static contexts (like a printed leaflet or a blog).

A.k.a. Stealth Address, Reusable Payment Code

Requirements of Reusable Address

- Sender cannot interact with the recipient
- Only recipient can spend the output
- No one else can link the transaction

Recap: Elliptic Curve Cryptography

Private key: k (a large number)

Public key: K = k * G (point multiplication)

Discrete logarithm problem: k –> K is easy;
K –> k is hard (computationally infeasible)

Distributive property: (k + j) * G = k * G + j * G

Private Keys in Bitcoin (Key Space)

The order of elliptic curves can be determined in polynomial time. We know that there are 115 792 089 237 316 195 423 570 985 008 687 907 852 837 564 279 074 904 382 605 163 141 518 161 494 336 $\approx 10^{77}$ possible private keys for Bitcoin.

For comparison, it is estimated that the observable universe contains around 10^{80} atoms.

Hierarchical Deterministic Wallets

New key pair for every tx. How to back them up?

Idea: Generate seed, derive all addresses from it.

Keypath: Navigate from seed (m/44'/0'/0'/0/0).

Hardened derivation: Only possible if you know the private key and the chain code (randomness).

Non-hardened derivation: Derive addresses from public key extended with associated chain code.

Watch-only wallet: Derive addresses from xpub.

Recap: Key Derivation (BIP32)

Calculate a pseudo-random but deterministic jump j from the previous to the next private key.

Extend each key with additional randomness r (known as chain code) to preserve your privacy.

Next private key: $k' = k + j$

Next public key: $K' = K + j * G$

Hardened derivation: $(j, r') = \text{hash}(r, \mathbf{k}, \text{index})$

Non-hardened derivation: $(j, r') = \text{hash}(r, \mathbf{K}, \text{index})$

Diffie-Hellman (DH) Key Exchange

How to derive shared secret over public channel?

Alice: private key k_A, public key $K_A = k_A G$

Bob: private key k_B, public key $K_B = k_B G$

They exchange their public keys, then compute the **shared secret** as $k_A K_B = k_A k_B G = k_B K_A$.

The first public-key protocol, published in 1976.

Jump with Non-Interactive DH

Instead of paying Alice directly to $A = aG$ with a being private key of Alice, Bob chooses random value b and computes $B = bG$ and $S = bA$.

By hashing S to $j = \text{hash}(S)$, Bob can send coins to Alice by locking them with the key $A' = A + jG$.

Once Alice learned B (see the next slide), she can calculate $S = aB$, $j = \text{hash}(S)$ and $a' = a + j$.

Notify Recipient about Payment

In order to spend the received coins, Alice needs to learn about B. This can be achieved either:

- **In-band**: Bob can encode B in an unspendable output (OP_RETURN) of the same transaction. Alice then needs to check the outputs of every single transaction on the blockchain and just try.
- **Out-of-band**: Bob can send B to Alice (together with the transaction ID) on another channel (like email). Disadvantage: Alice could lose message.

Separate Key to Scan Blockchain

In the in-band variant, Alice can let someone else scan the blockchain on her behalf without security risks by publishing different public keys for Diffie-Hellman key exchange and Bitcoin key derivation.

Problem: Whoever scans the blockchain needs to be able to perform the DH and calculate the jump. This requires knowledge of Alice’s DH private key.

Solution: Encode in address both $A = aG$ and $K = kG$. Then $S = bA = aB$, $j = hash(S)$ and $K' = K + jG$.