[Bitcoin-development] Stealth Addresses

Peter Todd pete at petertodd.org
Mon Jan 6 12:03:38 UTC 2014

- Previous message: [Bitcoin-development] Privacy and blockchain data
- Next message: [Bitcoin-development] Stealth Addresses
- **Messages sorted by:** [date] [thread] [subject] [author]

```
* Abstract

A Stealth Address is a new type of Bitcoin address and related
scriptPubKey/transaction generation scheme that allowers payees to
publish a single, fixed, address that payors can send funds efficiently,
privately, reliably and non-interactively. Payors do not learn what
other payments have been made to the stealth address, and third-parties
learn nothing at all. (both subject to an adjustable anonymity set)


* Acknowledgments

Credit goes to ByteCoin for the original idea.(1) Gregory Maxwell, Adam
Back, and others on #bitcoin-wizards contributed valuable input on the
implementation. Finally thanks goes to Amir Taaki for input on the
general idea of stealth addresses and use-cases.


* Background

Viewed generally a Bitcoin address is a mechanism by which a payee
instructs a payor to create a transaction such that the payee can spend
one or more of the transaction outputs. Of course, typically the address
is simply the hash of a pubkey, and the mechanism by which the funds are
made available to the payee is to simply create a scriptPubKey of the
following form:

    DUP HASH160 <pubKeyHash> EQUALVERIFY CHECKSIG

The problem however is address reuse: it is convenient for payees to
give one or more payor a single address and use it multiple times for
various purposes. This results in all those payments becoming trivially
linkable to each other by an attacker - a threat not only to the privacy
of the user, but also to all users of Bitcoin.(2)

BIP32 hierarchical deterministic wallets are frequently proposed as a
solution. Now an address is a chain code and the mechanism by which a
scriptPubKey is generated is to derive a one-time-use pubkey from that
chain code and some index i. However, this quickly runs into two main
```

problems:

1) Lack of privacy: While someone not in possession of the address can't
   link payments together, someone who is can.

2) State: If the index is not to be re-used wallets must either maintain
   per-address state, or somehow query for already used indexes, or
   somehow generate them in a sufficiently small range that the payee
   can recover the indexes. All these solutions are problematic.

A good example of where the BIP32-derivation solutions fails come up at
the Dark Wallet Hackathon where it was suggested by the author that for
the purpose of securing person-to-person payments OpenPGP public keys
and X.509 certificates be extended with a new user-id field containing a
Bitcoin address. Wallet software could then use either certificate
system to ensure funds were being sent to the intended recipients -
essentially a non-interactive way of solving what the BIP70 payment
protocol solves interactively. Of course, without stealth addresses the
scheme would likely have little or no privacy.

* Requirements

1) Generated scriptPubKey must be globally unique

2) Must be only spendable by payee

3) scriptPubKey and associated transaction must be indistinguishable to
   third-parties from other transactions in some anonymity set.

4) Method must be fully deterministic and funds recoverable from a
   wallet seed and blockchain data for both payee and payor.

5) Funds must be efficiently recoverable by payee with reasonable, and
   configurable, computation and bandwidth costs.

6) Must be compatible with CoinJoin/Must not leak information to payee
   about what txins were used to pay them.

7) Must be compatible with multisig-protected wallets.

8) Must not make assumptions about txin scriptSig form.

9) Must be possible to prove to third parties that payment was made in
   accordance to instructions without revealing any other information.

** Payment Reliability

Schemes for making payments by transmitting nonces to the recipient
through some other medium, such as Bitmessage, were discussed at the
Dark Wallet Hackathon. However using any medium but the blockchain
itself for the communication means that the reliability of the payment
getting to the recipient is less than that of a standard transaction.
For instance Bitmessage nodes only keep messages for two weeks. We

decided that anything less than reliable atomic transactions was unacceptable.

* Applying encryption to payments, simple explanation

Using Elliptic curve Diffie-Hellman (ECDH) we can generate a shared secret that the payee can use to recover their funds. Let the payee have keypair Q=dG. The payor generates nonce keypair P=eG and uses ECDH to arrive at shared secret c=H(eQ)=H(dP). This secret could be used to derive a ECC secret key, and from that a scriptPubKey, however that would allow both payor and payee the ability to spend the funds. So instead we use BIP32-style derivation to create Q'=(Q+c)G and associated scriptPubKey.

As for the nonce keypair, that is included in the transaction in an additional zero-valued output:

    RETURN <P>

The payee recovers the funds by scanning the blockchain for candiate P's in transactions, regenerating the scriptPubKey, and finally checking if any txouts in the transactions match. Note the close similarity of this technique to how the Bitmessage network functions - an initial implementation of the idea will find the Bitmessage code a suitable starting point.

* Trading off anonymity set size for decreased bandwidth/CPU

By taking advantage of prefix filters(3) we can choose a tradeoff between anonymity set size and bandwidth/CPU usage if the payee specifies that payments to them are to match some short prefix k. There are a few possibilities for how the prefix is to the applied - the most simple is if per-block indexes of scriptPubKeys are available:

    RETURN <k> <P>

Alternatively if per-block indexes of H(scriptPubKeys) are only available the wallet software can grind the scriptPubKey with nonce i until it matches the specified prefix:

    RETURN <i> <P>

Furthermore as symmetric ciphers are quite cheap we might as well hide the purpose of the OP_RETURN txout and encrypt the pubkey P using H(Q) as a symmetric key. This gives us a slightly larger anonymity set.

* Advantages of using a separate output

An alternative would be to either re-use a pubkey or signature nonce value from a transaction input, saving about 45 bytes per txout. An absolute minimum sized Bitcoin transaction is 166 bytes(4) so at best we have a 27% savings in tx fees, and more typically around ~15%. (modulo

mass-payments from a single txin)

However using an explicit prunable OP_RETURN output to store the pubkey rather than re-using one from a txin or txin signature has a number of advantages:

1) The txin's owned by the payor are not revealed to the payee. In fact, they could be held by a third-party who simply makes a transaction with the appropriate txouts on behalf of the payee.

2) Less information about the txouts is leaked. The statistical distribution of txouts remains unchanged - not possible in re-use schemes because they need to grind the payee scriptPubKey's for the sake of the prefix filters.

3) If required the nonce secret can be revealed to prove that a payment was made to a third-party, e.g. for dispute resolution.

* Bare CHECK(MULTI)SIG output alternative

An alternative with better efficiency could be to use bare OP_CHECK(MULTI)SIG outputs to hold the nonce pubkey - generally a second output is needed anyway for change. The most simple would be to use Jeff Garzik's OP_DROP proposal(5) for the prefix:

  <prefix> DROP n <pubkey>...<pubkey> m CHECKMULTISIG

  or

  <prefix> DROP <pubkey> CHECKSIG

The payor pubkey is in the *change* txout, and the payee's ECDH-derived pubkey in the other txout. By setting the prefix to be the same on both txouts and using the same basic scriptPubKey form the relationship of change and payment is still hidden; CoinJoin-using implementations can adopt even more sophisticated approaches.

If IsStandard() rules remain the same and using OP_DROP is impractical, we can also grind the change pubkey to match the prefix in a deterministic manner so the wallet can still be recovered from a seed. More costly, but maybe still acceptable for reasonably short prefixes. Either way the result is transactions that are actually smaller and cheaper than standard transactions, although without the advantage of pushing scriptPubKey size payment to the receiver. (a pity we didn't spend the extra time to adopt OP_EVAL)

A disadvantage is that revealing the nonce secret to prove a payment was made is more problematic - either the txout needs to be spent first, or we need a CHECKMULTISIG.

* Address format

To be decided. To support mulisig we probably want the ability to

specify n-of-m master pubkeys, using the nonce to generate derived ones.
For the single pubkey case the addresses will be a little longer than
standard Bitcoin addresses:

  s9KND3vfXjs3YqfZp86Acce3bM7Mhuptwh6mjeDnThsDei9Z2ZZcU

  vs.

  1LZn91ynrA6BCmoUKwnV3Ygk4FQMfPxLbg

1) ByteCoin, Untraceable transactions which can contain a secure message
   are inevitable, https://bitcointalk.org/index.php?topic=5965.0

2) Gregory Maxwell, Dark Wallet Certification discussions, also
   http://snowdenandthefuture.info/PartIII.html

3) Peter Todd, [Bitcoin-development] Privacy and blockchain data,
   http://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg03612.html

4) Bitcoin Wiki, Maximum transaction rate,
   https://en.bitcoin.it/w/index.php?title=Maximum_transaction_rate&oldid=36983

5) Jeff Garzik, Add small-data OP_DROP transactions as standard
   transactions, https://github.com/bitcoin/bitcoin/pull/1809

--
'peter'[:-1]@petertodd.org
000000000000000002861ee0919fc86990573ac360820766dc1b9ba580e5ccf7b6
-------------- next part --------------
A non-text attachment was scrubbed...
Name: signature.asc
Type: application/pgp-signature
Size: 685 bytes
Desc: Digital signature
URL: <http://lists.linuxfoundation.org/pipermail/bitcoin-dev/attachments/20140106/53d085cc/attachment.sig>

- Previous message: [Bitcoin-development] Privacy and blockchain data
- Next message: [Bitcoin-development] Stealth Addresses
- **Messages sorted by:** [date] [thread] [subject] [author]

More information about the bitcoin-dev mailing list