Homomorphic Mini-blockchain Scheme

B.F. França

24 April 2015

Abstract

We create a new cryptocurrency scheme based on the mini-blockchain scheme and homomorphic commitments. The aim is to improve the mini-blockchain by making it more private. We also make a comparison of Bitcoin and our scheme regarding their ability to resist blockchain analysis.

1 Introduction

Bitcoin [6] is a groundbreaking concept which has already made its mark into the world, although it is only 5 years old. Despite all its merits, Bitcoin still has major flaws which impede its adoption as a mainstream currency, namely:

Scalability: Bitcoin at this moment is only able to handle 7 transactions per second (tps), a far cry from the 10,000-100,000 tps required for it to become a global currency.

Speed: Bitcoin blocks are mined only at 10 minutes intervals. Bitcoin cannot compete with existing payment systems (Visa, Paypal, etc) unless it can make reasonably secure and fast transactions.

Privacy: All transactions are public and unencrypted in the blockchain so it is very easy to follow transactions or to find the balance associated with a given address. The current suggested practices to improve privacy (i.e. don't reuse addresses, use mixing services) are inefficient, difficult to implement and/or require third-party trust.

In the last few years there have been an explosion of cryptocurrencies and related research papers trying to solve one or several of the above problems. One of these was the mini-blockchain scheme which modified the original blockchain in order to reduce its size and support increased block size. This paper attempts to build upon the mini-blockchain scheme and modify it to achieve greater privacy.

2 Building blocks

2.1 Mini-blockchain

The mini-blockchain [2] (MBC) was designed by J.D. Bruce as an improvement to the original blockchain created by S. Nakamoto. The main innovation was the introduction of the "account tree", which is basically a balance sheet storing the balance of every account. With this change, transactions no longer need to be stored forever in the blockchain, only the most recent transactions and the current account tree. The mini-blockchain is therefore much more scalable than the original blockchain since the mini-blockchain only grows when new accounts are created.

The mini-blockchain consists of 3 components:

- Account tree
- Transaction tree
- Proof chain

We will now succinctly describe each one. Firstly, the account tree is a Merkle tree of all the accounts in a given block; each account being a data block with an address and a balance (it can have more data fields, if necessary). Secondly, the transaction tree is a Merkle tree of all transactions in a given block, each transaction representing a change to a number of accounts. Finally, the proof chain is simply a chain of blocks where each block contains a nonce, the top hash of the account tree and of the transaction for that block and the hash of the previous block. Essentially, it is the headers of a normal blockchain.

The mini-blockchain operates very similarly to the regular blockchain. With each block the client nodes submit their transactions to the network. Following this, each miner, independently, verifies the correctness of the transactions and creates a transaction tree with the correct ones. Each miner also modifies the account tree to reflect the changes done by the transactions. Finally, like in a normal blockchain, the miner creates the current block in the proof chain by trying different nonces until the hash of the block has a given number of zeros. If a miner can find such a nonce, he can submit it to the network to be included in the mini-blockchain. Unlike a regular blockchain, nodes only need to keep a finite number of account trees and transaction trees so, after a new block is created, the account tree and the transaction tree of an older block is discarded. Only the proof chain needs to be stored in its entirety.

This paper only attempts to change how accounts and transactions are coded into the mini-blockchain, everything else is going to remain intact.

2.2 Elliptic curve Pedersen commitment scheme

The elliptic curve Pedersen commitment scheme is a variant of the Pedersen commitment scheme [7] based on elliptic curve cryptography. Let $\mathbf{F}_p$ be the group of elliptic curve points, where p is a large prime. Also, let G and H be

two random points in $\mathbf{F}_p$. To commit to a value $x \in \mathbf{Z}_p$, we pick a random integer $r \in \mathbf{Z}_p$ and calculate the commitment as

$$Com(x, r) = xG + rH$$

To open the commitment we simply reveal the values x and r. EC Pedersen is also additively homomorphic, so it has the following properties:

$$Com(x + y, r_x + r_y) = Com(x, r_x) + Com(y, r_y)$$

$$Com(k \cdot x, k \cdot r) = k \cdot Com(x, r)$$

2.3 Elliptic curve hashed Elgamal encryption

Elliptic curve hashed Elgamal is a variant of standard Elgamal encryption [4] that allows us to encrypt arbitrary messages using elliptic curves. However, it is not homomorphic like standard Elgamal. To use this scheme we need an elliptic curve group of prime order, $\mathbf{F}_p$, and an hash function, $h : \{0,1\}^* \to \{0,1\}^s$. s denotes the size of the hash function output in bits. The scheme then works as follows:

Key generation: Pick a random integer $x \in \mathbf{Z}_p$. The public key is $Q = xP$, where P is the base point of the elliptic curve. The private key is x.

Encrypting: Pick a random integer $r \in \mathbf{Z}_p$. For a message m, the encryption is the tuple $(R, c) = (rP, m \oplus h(rQ))$. Where $\oplus$ is the xor operator.

Decrypting: For a cyphertext (R, c), calculate the message as $m = c \oplus h(xR)$.

Note that the size of the message is limited to s bits.

2.4 Boneh-Lynn-Shacham signature scheme

The Boneh-Lynn-Shacham (BLS) signature scheme [3] is a digital signature scheme based on bilinear pairings in elliptic curves. The BLS signature scheme produces very short signatures and is quick to verify. Let $e : \mathbf{F} \times \mathbf{F} \to \mathbf{G}$ be a bilinear pairing, $\mathbf{F}$ and $\mathbf{G}$ elliptic curve groups of prime order p and P the base point in $\mathbf{F}$. Also, let $H : \{0,1\}^* \to \mathbf{F}$ be an hash function that hashes bit strings to elliptic curve points (see Annex 1). The BLS signature scheme consists of the following three operations:

Key generation: $x \leftarrow_R \mathbf{Z}_p, Q = xP$. The public key is Q and the private key is x.

Signing: For a message m, the signature is $\sigma = xH(m)$.

Verifying: For a signature σ and message m, check if $e(\sigma, P) = e(H(m), Q)$. Accept if true.

Where $x \leftarrow_R \mathbf{Z}$ means "to choose at random x from $\mathbf{Z}$". This signature scheme also has variants for threshold signatures, multisignatures and blind signatures [1].

2.5 Non-interactive zero knowledge range proof

A non-interactive zero knowledge (NIZK) range proof is needed to prove that a given committed number x belongs to a range $[a; b]$. The particular NIZK range proof that will be used is called binary decomposition range proof [5] and it proves that a Pedersen committed number is in a range of the form $[0; 2^n[$. We will be using an optimized version due to Adam Back. Given a commitment $E = xG + rH$ to x, an hash function $h : \{0;1\}^* \to \{0;1\}^\delta$ and a pseudo-random number generator $PRNG : \{0;1\}^\delta \times \mathbf{N} \to \mathbf{Z}_p$ we construct a proof that $x \in [0; 2^n[$ by running the following protocol:

1. Commit to the n individual bits of x; $E_i = x_i G + r_i H, x_i \in \{0;1\}, r_i \leftarrow_R \mathbf{Z}_p, i \in \{0, ..., n-1\}$.
2. $s = h(E), u = h(E_0, ..., E_{n-1})$
3. For all i:
   (a) If $x_i = 0$: $w_i \leftarrow_R \mathbf{Z}_p, A_i = w_i H$
   (b) If $x_i = 1$: $t_i \leftarrow_R \mathbf{Z}_p, c_{1i} = PRNG(s, i), A_i = t_i H - c_{1i} E_i$
   (c) $c_i = h(A_i)$
   (d) If $x_i = 0$: $c_{1i} = PRNG(s, i), c_{2i} = c_i - c_{1i}, t_i = w_i + r_i c_{1i}, B_i = t_i H - c_{2i}(E_i - G)$
   (e) If $x_i = 1$: $c_{2i} = c_i - c_{1i}, w_i = t_i - r_i c_{2i}, B_i = w_i H$
4. $v = h(B_0, ..., B_{n-1})$
5. The proof is the tuple $(u, v, A_0, r_0, ..., A_{n-1}, r_{n-1})$.

The range proof can then be verified by anyone who runs this protocol:

1. $s = h(E)$
2. For all i:
   (a) $c_{1i} = PRNG(s, i), c_i = h(A_i), c_{2i} = c_i - c_{1i}$
   (b) $E_i = \frac{1}{c_{1i}}(t_i H - A_i), B_i = t_i H - c_{2i}(E_i - G)$
3. Check if $u = h(E_0, ..., E_{n-1})$, $v = h(B_0, ..., B_{n-1})$ and $E = \sum_{i=0}^{n-1} E_i 2^i$. Accept if true.

2.6 Zero knowledge equality proof

This is a ZK proof that the following relation holds, $\sum_i a_i - \sum_j b_j = c$ for committed numbers a_i and b_j and for public value c. It is a simple application of the homomorphic property of the Pedersen commitment scheme. Given commitments $A_i = a_i G + r_i H$ and $B_j = b_j G + s_j H$ and a number c, the proof is simply the value $t = \sum_i r_i - \sum_j s_j$. Anyone can verify the proof by checking that $cG + tH = \sum_i A_i - \sum_j B_j$.

3 Description

Conceptually the homomorphic mini-blockchain (HMBC) scheme is composed of 4 types of objects:

Fixed key pair: Two elliptic curve key pairs created by a user and kept by him for as long as he wants. The pair of public keys, which is called a fixed address, is needed in order to make payments to the user.

Temporary key pair: A elliptic curve key pair that is derived from a fixed key pair. It is created during a transaction by the payer from the fixed address of the payee and is only valid for that transaction. The payee needs the temporary private key to use the coins that he received. The public key is called a temporary address.

Receipt: It proves that a given temporary address owns a determinate number of coins. It can only be used once.

Transaction: It authorizes the movement of coins from some temporary addresses to others. In the process it destroys and creates receipts.

The blockchain in our scheme is the mini-blockchain with the difference that instead of "accounts" we have "receipts". Although a receipt can be thought of as a single-use account. Otherwise, it is exactly the same. In the next sections we will describe in more detail each of these objects.

3.1 Fixed and temporary key pairs

A fixed key pair is created by the following protocol:

1. $x, y \leftarrow_R \mathbf{Z}_p$
2. $P = xG, Q = yG$

Where G is the base point in a elliptic curve group of prime order p with an efficient bilinear pairing. (P, x) is the "signing" key pair and (Q, y) is the "scanning" key pair. The fixed address is the tuple (P, Q). A temporary key pair is derived from the signing key pair in the following manner:

1. $t \leftarrow_R \mathbf{Z}_p$
2. $P' = tP, x' = tx$

The temporary address is the point P'. Note that the temporary private key can only be calculated if one has the signing private key.

3.2 Receipt

A receipt is the following tuple:

$$(h(P'\|B), wG, \phi\|b\|r_s \oplus h(wQ))$$

We can divide it into the "head" and the "body".
The head is $h(P'\|B)$, where $h(\cdot)$ is an hash function, $\|$ is the concatenation operator, P' is a temporary address and $B = bG + r_bH, r_b \leftarrow_R \mathbf{Z}_p$ is a Pedersen commitment to the value (number of coins) b of the receipt. So the head, through the temporary address and the value commitment, contains the information that a given fixed address owns a given number of coins.
The body is $(wG, \phi\|b\|r_s \oplus h(wQ))$. It is simply an elliptic curve hashed Elgamal encryption of the message $\phi\|b\|r_s$ using the scanning public key Q and randomizer w. The message consists of a "padding" of zeros ϕ, the value b and a random seed r_s. This random seed is used with a pseudo-random number generator $PRNG : \{0;1\}^{128} \times \mathbf{N} \rightarrow \mathbf{Z}_p$ to create the temporary address and the value commitment; $t = PRNG(r_s, 0), r_b = PRNG(r_s, 1)$. So an user can use the body of a receipt to:

- Quickly check if he is the owner of a receipt by decrypting it and seeing if it starts with the padding ϕ.
- Get the information needed to open the value commitment B, namely b and $r_b = PRNG(r_s, 1)$.
- Calculate his temporary private key, $x' = tx = PRNG(r_s, 0)x$.

We remark that the "scanning" key pair (Q, y) is only used to encrypt/decrypt the bodies of receipts, it not used anywhere else in this scheme.

3.3 Transaction

A transaction is composed by the following data fields:

- For every input receipt i:
  - Temporary address, $P'_i = x'_iG$
  - Value commitment, $B_i = b_iG + r_{bi}H$
- For every output receipt j:
  - Temporary address, $P'_j = x'_jG$
  - Value commitment, $B_j = b_jG + r_{bj}H$
  - Receipt body, $(w_jG, \phi\|b_j\|r_{sj} \oplus h(w_jQ_j))$
  - NIZK range proof, a proof that $b_j \in [0; 2^\delta[$ (see section 2.5)
- Transaction fee, τ

- ZK equality proof, a proof that $\sum_i b_i - \sum_j b_j = \tau$ (see section 2.6)
- Message, an optional field to write arbitrary messages
- Time-lock, an optional field that states that the transaction is only valid after a given time
- For every input receipt i:
  - Signature, $\sigma_i = x'_i H(m)$

Where $2^\delta - 1$ is the maximum value of a receipt. The range proofs are needed in order to prevent the values in the equality proof from "wrapping around", since the equality proof only works over $\mathbf{Z}_p$ and we need to prove the relation $\sum_i b_i - \sum_j b_j = \tau$ over $\mathbf{Z}$. We will now describe briefly how a transaction is created and processed.
Suppose Alice wants to pay b coins to Bob, that she knows Bob's fixed address (Q_B, P_B) and that her fixed address is (Q_A, P_A). First she will choose a receipt(s) with a value greater than b plus the desired transaction fee τ, calculate the corresponding temporary key pair (P'_A, x'_A) and value commitment B (remember she can do this by decrypting the body of the receipt) and write the temporary address and the value commitment in the transaction. Secondly, she will pick a random integer $r_s \in [0; 2^{128}[$ that she will use as a seed to calculate Bob's temporary address, $P'_B = tP_B = PRNG(r_s, 0)P_B$, and the corresponding value commitment, $B_B = bG + r_b H = bG + PRNG(r_s, 1)H$. She will also pick a random integer $w \in \mathbf{Z}_p$, use it together with Bob's scanning public key Q_B to create the receipt's body, $(wG, \phi \| b \| r_s \oplus h(wQ_B))$, and create the NIZK range proof that $b \in [0; 2^\delta[$. Then she will do all this again but to create a receipt for herself in the value of the change amount. Thirdly, she will write the transaction fee and calculate the ZK equality proof to guarantee that her inputs match the outputs plus the transaction fee. Optionally, she may write a message or set a time-lock on the transaction. Finally, she will sign the transaction with her temporary private key using the BLS signature scheme and broadcast the transaction.
To process a transaction, a miner first sees if there is an active time-lock or if the transaction fee is insufficient. If that's not the case, then he hashes the input receipt's temporary address and value commitment and searches for a receipt with the same hash (as a "head"). If he finds it, he will then verify the correctness of the NIZK range proofs, the ZK equality proof and the BLS signatures. Assuming all are valid he adds the transaction to his transaction tree for the current block and updates the receipt tree (by deleting the input receipts and adding the output receipts). If he succeeds in mining the block, the transaction has made its way into the blockchain.
There exists also another transaction, called a minting transaction, used to claim a block's reward. Suppose the block reward is b and that the miner's fixed address is (Q, P). A minting transaction consists of only these data fields:

- Temporary address, $P' = x'G$

- Value commitment, $B = bG + r_bH$
- Value commitment randomizer, r_b
- Receipt body, $(wG, \phi \| b \| r_s \oplus h(wQ))$

The value commitment randomizer r_b is needed in order to verify that the value commitment hides the correct value b. When mining, the miner only needs to include this transaction and the corresponding receipt in the block that he is mining.

4 Extra features

In the last section the basic HMBC scheme was introduced, in this section some additions to the scheme will be presented that will allow us to have minimum output values, multisignature addresses and blind signatures.

4.1 Minimum output value

We can use modify the NIZK range proof in the transaction protocol in order to impose a minimum value for every output. Suppose the minimum output value is c, that Alice is creating a transaction where one of the ouput values is b and the corresponding commitment is $B = bG + r_bH$. Alice can simply recalculate her commitment $B' = B - cG = (b - c)G + r_bH$ and then run the NIZK range proof on the new commitment thus proving $b \in [c; 2^\delta + c[$. Note that the upper bound of the range also changed by $+c$, but that is not a problem since we don't require the upper bound to be tight.
This feature can be used to prevent "dust" creation in the blockchain.

4.2 Multisignature addresses

As with Bitcoin, multisignature addresses in HBMC scheme allows n users to create an address such that any $t \in [1; n]$ of them can sign transactions. The difference is that while in Bitcoin multisignature addresses are simply a collection of n public keys and the threshold t, multisignatures addresses in the HMBC scheme are equal to regular addresses where the secret key is shared between the n users. So, multisignature addresses in our scheme are indistinguishable from regular addresses and support the creation of temporary addresses.
We differentiate between 3 types of multisignature addresses; $t = n$, $1 < t < n$ and $t = 1$. We will describe only the first two cases since the $t = 1$ case is trivial. Also, we will discuss shared signing key pairs, for threshold signing of transactions, and shared scanning key pairs, for threshold decryption of receipts.

4.2.1 t=n

Suppose that each of the n users have a key pair $(P_i, x_i), i \in [1; n]$. They can create a shared public key by summing all their individual public keys,

$$P = \sum_{i=1}^{n} P_i$$

The corresponding shared secret key will be $x = \sum_{i=1}^{n} x_i$ but it will be unknown to all n users. To sign a message m, each user signs with his own private key,

$$\sigma_i = x_i H(m)$$

and then sends the signature to a dealer (does not need to be trusted and can be one of the users) who calculates the shared signature

$$\sigma = \sum_{i=1}^{n} \sigma_i = \sum_{i=1}^{n} x_i H(m) = xH(m)$$

To decrypt a cyphertext $(R, s) = (wG, m \oplus h(wP))$, each user calculates,

$$C_i = x_i R$$

and then sends it to a dealer who decrypts the cyphertext

$$m = s \oplus h\left(\sum_{i=1}^{n} C_i\right) = s \oplus h\left(\sum_{i=1}^{n} x_i R\right) = s \oplus h(xR)$$

4.2.2 1<t<n

The n users create a (t, n) shared key pair by using the GJKR distributed key generation protocol [8] (see annex 2). This will require 2 rounds of synchronous communication and will create a key that is divided among the users using (t, n) Shamir's secret sharing. So for any set $S \subset \{1, ..., n\}$ of users, such that $|S| = t$, the secret x can be calculated by the following formula

$$x = \sum_{i \in S} l_i x_i \ , \quad l_i = \prod_{j \in S, j \neq i} \frac{j}{j - i}$$

Where x_i is secret share of user i. When signing a message m, each of the t users sign with his own private key,

$$\sigma_i = x_i H(m)$$

and sends it to a designated user to construct the signature,

$$\sigma = \sum_{i \in S} l_i \sigma_i = \sum_{i \in S} l_i x_i H(m) = xH(m)$$

Decrypting works similarly. For a cyphertext $(R, s) = (wG, m \oplus h(wP))$ each user calculates,

$$C_i = x_i R$$

and then sends it to a dealer who decrypts the cyphertext

$$m = s \oplus h\left(\sum_{i \in S} C_i\right) = s \oplus h\left(\sum_{i \in S} x_i R\right) = s \oplus h(xR)$$

For the special case 2-of-3 there is a simpler protocol based on the Diffie-Hellman key exchange that only requires one round of communication. Imagine that each one of the users $i \in \{0, 1, 2\}$ has a key pair (Q_i, y_i) and that they know each others public keys. Then they can create a shared secret by running the following protocol:

1. For each user i:
   (a) Calculate $x_{i,i-1} = h(y_i Q_{i-1})$ and $x_{i,i+1} = h(y_i Q_{i+1})$.
   (b) Send $P_{i,i-1} = x_{i,i-1} G$ to user $i+1$ and $P_{i,i+1} = x_{i,i+1} G$ to user $i-1$.
   (c) After receiving $P_{i-1,i+1}$ from user $i-1$ and $P_{i+1,i-1}$ from user $i+1$, check if $P_{i-1,i+1} = P_{i+1,i-1}$. Accept if true.
   (d) Calculate $P = P_{i,i-1} + P_{i,i+1} + P_{i-1,i+1}$.

The subscripts $i-1$, i and $i+1$ are all calculated modulo 3. $h : \{0;1\}^* \to \{0;1\}^{256}$ is an hash function. The shared secret is $x = x_{i,i-1} + x_{i,i+1} + x_{i-1,i+1}$ and of course $P = xG$. The signing and decrypting processes are equal to the $t = n$ case.

4.3 Blind signatures

Imagine Alice has a message m that she wants signed by Bob, who has key pair (P, x), but she does not want Bob to know that he is signing m. To do this Alice first calculates the message hash $H(m)$ and "blinds" it,

$$H'(m) = rG + H(m), r \leftarrow_R \mathbf{Z}_p$$

Then she gives it to Bob who will sign it,

$$\sigma' = xH'(m)$$

Finally Alice "unblinds" the signature,

$$\sigma = \sigma' - rP = x(rG + H(m)) = rP + xH(m) - rP = xH(m)$$

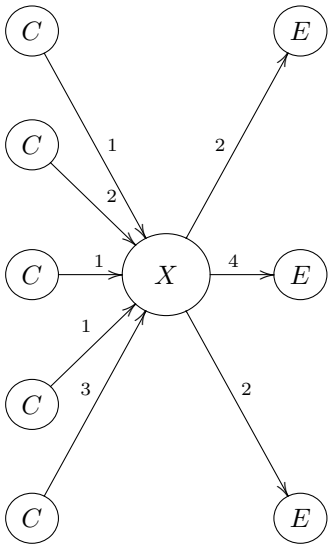
This feature can be used to create "wallets" that are both private and secure. For example, a 2-of-2 multisignature address with the scanning key and one signing key belonging to the client while the other signing key belongs to the server. The server adds security by only signing transactions after verifying that they were created by the client (ex: two-factor authentication, know-your-customer,etc) while keeping the client's privacy by blind signing the transactions.

5 Privacy

In this section we will try to determine the level of privacy in HMBC scheme. In order to do this we are going to present some common blockchain analysis techniques and show how the HBMC fares against them.

5.1 Address reuse

The simplest and most effective analysis technique is finding addresses that are reused. Many times, because of carelessness or practicality, a Bitcoin address is used similarly to a bank account. To exemplify the power of this technique, imagine that the following transaction graph is of a given company X receiving payments from its customers C and paying salaries to its employees E,

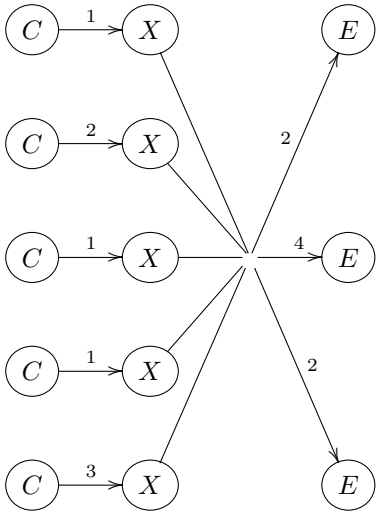


Anyone with access to the blockchain can see this graph and deduce from it:

- The balance of the company at any time.
- How much it receives from each customer.
- How much it pays each employee.
- If any of the customers or the employees also reuses his address, all the transactions between them can be linked.

To make things worse, most people who reuse addresses also post online that they own those addresses thereby linking their identities with all the transactions they make.
The HMBC scheme enforces the use of single-use addresses while giving the

users the option of having a fixed address. The above scenario without address reuse would look like this,

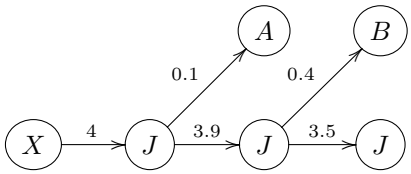


Now that each transaction has its own temporary address no one can deduce company's X balance or link several customers/employees transactions. More importantly, since the temporary addresses are not publicized as belonging to a given person, no one can link them to the identity of their owner.

5.2 Transaction value

The values of all transactions are public and that can be a problem. Even though the temporary addresses impede an observer from linking transactions to identities, the persons involved in a transaction will usually know the identities of the other persons involved in the same transaction. For example in the scenario we described earlier it is reasonable to assume that the employees know each other, after all they work together. Any of them can look at the transaction and see that one of them received more than the others, although they can't deduce from the transaction graph who it was.
A second concern about public transaction values is that unusual or big values stand out and can be used to filter the transactions in the blockchain. For example, if someone knows that a large transaction occurred at a given hour or if someone knows the exact value of a oddly valued transaction.
But the most effective use of the transaction value is in guessing the change address in a given transaction. Imagine this situation, Joe just received his salary from company X and shops at two different stores A and B. The transaction

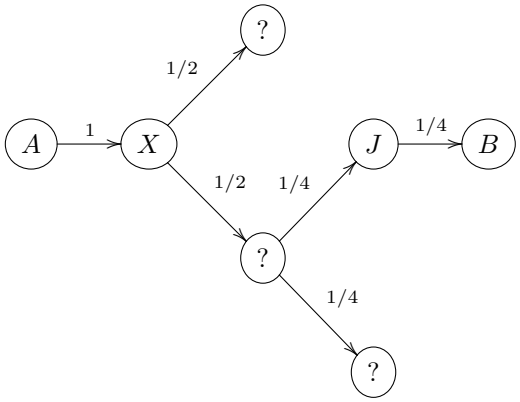
graph would look like this,



When seeing the transaction values it is reasonable to assume that the bigger output is the change address. And it gets more obvious with each transaction Joe makes because it forms a longer and longer chain. The problem here is any person who receives a payment from Joe, like A and B, can then look at the chain of payments and see how many coins Joe still has, how many he spent until now and the specific payments he made. They can even continue following the chain as Joe continues making purchases. They just won't know to whom did Joe pay.
There are also situations where the most reasonable would be to guess that the smaller output is the change address. The point is that transaction values are very valuable when trying to determine the change address.
All the transaction values (except the transaction fee) in the HBMC are encrypted and so they cannot be used for any of the above purposes. It is not possible to see the values for the other inputs/outputs in a transaction that you are a part of, to filter the transactions in the blockchain based on their value or to guess the change address any better than random.

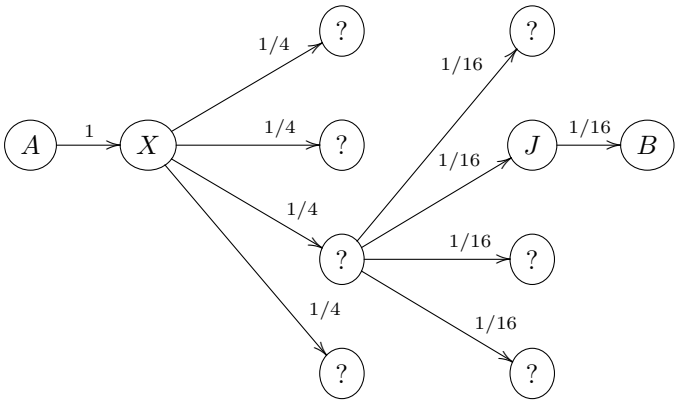
5.3 Flow

Flow analysis consists of simply following the "flow" of coins through different addresses. Since there is no address reuse or public transaction values, this technique is much less effective in the HBMC scheme than in Bitcoin. But it still can be used to deanonymize an user. Imagine the following scenario, user A makes a payment to user X and later Joe makes a payment to user B.



The arrow labels denote the probability that the destination address belongs to user X. To calculate this probability it is assumed that there is always exactly

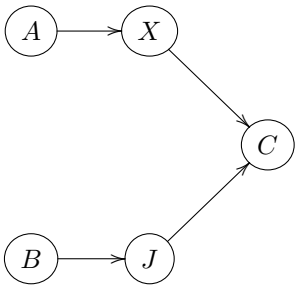
one change address.
If user A and user B collude, they know that there is a 25% chance that user X is Joe. If it is true, A has learned the identity of X and B has learned that Joe received a payment from A.
We can't fully protect against this attack but we can make it much less effective. To do this user X only needs to add a few "dummy" outputs to each transaction he makes. A "dummy" output is an output that is included in a transaction only to increase the number outputs. A "dummy" output should have a low value, but bigger than the transaction fee so that it does not became dust, and a random existent address, it is enough to pick a random fixed address from the internet. The transaction graph will then look like this,



By introducing just two "dummy" outputs in each transaction, A and B now only have 6.25% certainty that user X is in fact Joe. The number of "dummy" outputs can be increased to make that certainty arbitrarily low.

5.4 Input aggregation

The final technique is a very simple heuristic, whenever there are multiple inputs to a transaction it is assumed that they are owned by the same person. It can be used to deanonymize a transaction in a similar manner to flow analysis. Imagine user A makes a payment to user X, that user B makes a payment to Joe and that X/Joe makes a payment to another user C using those inputs.



If user A and user B collude they will discover that user X and Joe are the same person. Unfortunately, the only way to defend against this attack in the HBMC scheme is the same as in Bitcoin, to use Coinjoin transactions. However, Coinjoin transactions will be more efficient in the HBMC scheme because of the transaction values being encrypted.

6 Conclusion

Starting from the mini-blockchain scheme we constructed a cryptocurrency scheme that is, at the same time, more private and more scalable than Bitcoin. It is also more practical for the end-user because of the use of fixed addresses, scanning keys, native multisignature support, blind signing, etc. We hope that this scheme incites the appearance of innovative services and, perhaps, better cryptocurrencies.

References

[1] Alexandra Boldyreva. Efficient threshold signature, multisignature and blind signature schemes based on the gap-diffie-hellman-group signature scheme. In *Proceedings of PKC 2003, Volume 2567 of LNCS*, pages 31–46, 2003.

[2] J. D. Bruce. The mini-blockchain scheme, 2014. `http://www.cryptonite.info/files/mbc-scheme-rev2.pdf`.

[3] Ben Lynn Dan Boneh and Hovav Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17:297319, 2004.

[4] Taher ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[5] Wenbo Mao. Guaranteed correct sharing of integer factorization with off-line shareholders. In *Proceedings of PKC 1998*, pages 60–71.

[6] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `https://bitcoin.org/bitcoin.pdf`.

[7] Torben Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology CRYPTO 1991*, pages 129–140.

[8] Hugo Krawczyk Rosario Gennaro, Stanisaw Jarecki and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology - EUROCRYPT 1999*, pages 295–310.

7 Annexes

7.1 Annex 1: Map-to-curve hash function

Let E be a elliptic curve $y = x^3 + ax + b$ over $\mathbf{Z}_p$ and $h : \{0;1\}^* \to \mathbf{Z}_p$ an hash function. Then we can hash arbitrary messages m into E by following this protocol:

1. Calculate $e = h(m)$.
2. For $i = 0$ to $k - 1$:
   (a) Calculate $x = e + i$.
   (b) Check if $x^3 + ax + b$ is a quadratic residue in $\mathbf{Z}_p$. If true, end the loop.
3. Return the point $(x, \sqrt{x^3 + ax + b})$

7.2 Annex 2: GJKR distributed key generation protocol

Let $\mathbf{F}_p$ the group of prime order p formed by the points of a elliptic curve over $\mathbf{Z}_q$. Also, let G and H be two points in $\mathbf{F}_p$. Then, n users can create a key shared between with a threshold of t using the following protocol:

1. Generating x:
   (a) For each user P_i:
      i. Choose two random polynomials $f_i(z)$ and $g_i(z)$ of degree t over $\mathbf{Z}_p$,

$$f_i(z) = a_{i0} + a_{i1}z + \cdots + a_t z^t \quad g_i(z) = b_{i0} + b_{i1}z + \cdots + b_t z^t$$

      ii. Let $z_i = a_{i0}$. Calculate and broadcast $C_{ik} = a_{ik}G + b_{ik}H, k \in \{0, ..., t\}$.
      iii. Calculate $s_{ij} = f_i(j)$ and $s'_{ij} = g_i(j)$ for all $j \in \{1, ..., n\}$. Send $s_{ij} = f_i(j)$ and $s'_{ij} = g_i(j)$ privately to each user P_j.
      iv. After receiving s_{ji} and s'_{ji} for all $j \in \{1, ..., n\}$, check the following relation:

$$s_{ji}G + s_{ji}H = \sum_{k=0}^{t} i^k C_{jk}\ ,\ \forall j \in \{1, ..., n\}$$

      If false for any given j, broadcast a complaint against user P_j.
   (b) Any user P_i who receives a complaint from another user P_j broadcast the values s_{ij} and s'_{ij}.
   (c) Each player marks as disqualified another player that either:
      i. Received more than t complaints.

ii. Answered to a complaint with wrong values.

(d) Each player builds the set of non-disqualified users, Ω.

(e) Each player P_i calculates his share of the secret:

$$x_i = \sum_{j \in \Omega} s_{ji} \mod p$$

The secret is $x = \sum_{j \in \Omega} z_j \mod p$.

2. Generating $Q = xG$:

(a) For each user $P_i \in \Omega$:

i. Calculate and broadcast $A_{ik} = a_{ik}G$, $\forall k \in \{0, ..., t\}$.

ii. After receiving A_{jk} for all $j \in \{1, ..., n\}$ and $k \in \{0, ..., t\}$, check the following relation:

$$s_{ji}G = \sum_{k=0}^{t} i^k A_{jk}, \ \forall j \in \{1, ..., n\}$$

If false for any given j, broadcast a complaint and the values s_{ji} and s'_{ji}.

(b) Whenever a user P_i receives a valid complaint, the other users jointly calculate z_i and $f_i(z)$.

(c) Each user $P_i \in \Omega$ calculates $Q = \sum_{i \in \Omega} A_{i0}$.