MAVEPAY, A NEW LIGHTWEIGHT PAYMENT SCHEME FOR PEER TO PEER CURRENCY NETWORKS

SERGIO DEMIAN LERNER

ABSTRACT. In this paper we propose a new payment scheme based on the MAVE digital signature protocol, for use in peer to peer currency networks based on a block chain. The proposed payment scheme requires less resources for the users than Bitcoin and can be used to create a truly lightweight peer to peer currency network that stands 700 payments/second, and 5 million new user accounts/year, on an average home computer with an average Internet connection, at a fixed cost per client of less than 13 USD/month (as of 2011, electricity, bandwidth and PC amortization costs included, in the U.S.).

1. Introduction

Bitcoin is a decentralized electronic currency system proposed as an alternative to current government-backed currencies [11]. It was created by Satoshi Nakamoto in 2008 [7] along with the first reference implementation. Bitcoin protocol relies on the public validation of transactions, and assumes every end-user will always be able to handle the burden of validating every transaction. Nevertheless, Bitcoin protocol was not specifically chosen to put end-user resources consumption down to a minimum. Transactions messages are unnecessary long and the storage required is unnecessary high. Most notably, the choice of signature scheme (ECDSA) favors short signatures over verification time (RSA signatures are much faster to verify, although longer) [4]. This choices have imposed very tight limitations on the maximum size of the network [1]. The key limiting factors are bandwidth usage, chain size and CPU processing time. Protocol upgrades, as proposed numerous times in forums, may put the limits a little forward, but cannot increase the network capacity by an order of magnitude. CPU processing time limit imposed by the chosen ECDSA curve is the harder to surpass, without redesigning the whole protocol. There have been many proposals for micropayment systems that rely on one time signatures (OTS) that rely on hash chains to lower the CPU consumption requirement [9] for payees and payers. Nevertheless these protocols rely on a central trusted authority both to issue money and to validate the signatures. Recently the digital signature protocol MAVE-3 has been proposed [6]. We present a new payment scheme based on MAVE-3 and extend the MAVE-3 signature to hold the payload required to be applicable to a peer to peer currency network. We also replace the aggregator by competing miners, as in Bitcoin. MAVEPAY/MAVE-3 use commitments to construct signatures. Commitment based ownership was proposed in parallel to this work in [5], although that proposal is incomplete (it cannot efficiently link transaction messages together and so it avoids dealing with the delay attack or the command flooding attack).

Key words and phrases. Bitcoin P2P digital e-cash electronic currency.

A general principle of an efficient free market is that it should provide consumers with a variety of options from which they can choose the quality and price that best suits their needs. This principle can be applied to P2P currency networks as the Least Required Security (LRS), when we achieve lower costs (for the user and for the network) for lower security. Instead of providing a digital signature algorithm with a fixed key-length (e.g. ECDSA secp256k1) we provide a digital signature algorithm with selectable security thresholds such that the lower the key size, the lower the resource requirements for the network in terms bandwidth and storage space. Users are persuaded of using the key size so the cost of breaking the algorithm is higher than the funds stored. The incentive to use the least required key-size is monetary, since transactions on accounts with higher key-sizes pay higher fees because of their longer sizes.

Because each created account consumes resources, we want to keep the number of created accounts to a minimum. We could easily extend MAVE-3 signatures to allow simultaneously signing with multiple public keys, and so accept the simultaneous payment from multiple source accounts, but this would incentive the behavior of having multiple accounts. Still, if a user wants to make a payment using money from different accounts, payments can be first consolidated in a single account with multiple payments.

Table 1 show a comparison between Bitcoin (in its current implementation) and MAVE-3.

Property	Bitcoin	MAVE-3
Maximum transactions per second (*1)	10	700
Average block confirmations per payment	6	at least 18
Underlying security	ECDSA / SHA-2	truncated hash function
Fees in every message	Yes	No (*2)
Issue Payments during chain competition	Yes	No
Approximate client cost per month(*3)	13 USD	13 USD

TABLE 1. Bitcoin and MAVE-3 comparison

- *1 For an average home PC, using no more than 15% of CPU and 50% of bandwidth, and replacing the hard drive not before 2 years.
- *2 POW or similar defense required for some messages
- *3 As of 2011, electricity, bandwidth and PC amortization costs in the U.S. considered.

2. Key-sizes and the free market

Moore Law predicts the doubling of the processing power of users of the P2P currency every two years, but it also predicts the doubling of the attackers capabilities during the same period. A system that conforms to the LRS principle must be able to allow users to increase account security gradually, allowing to establish longer private key sizes to accommodate the growth in computing power. Bitcoin clearly does not conform to the LRS principle, since it forces a 2 cents transaction to use the same global resources than a 1000K coins transaction. To achieve the LRS principle we want to enable a "free market" in terms of security. As each transaction

pays a fee and implies a cost to the network to handle the transaction verification, transfer and storage, then the ideal scheme would create a market where fees and network cost are proportionally related, living a gap for miners revenue.

3. Multiple Digital Signature Algorithms

Any durable cryptocurrency must withstand the partial breakage of a cryptographic algorithm used. In such case, user would gradually start moving the funds from accounts whose security is questioned to accounts that rely on still unbroken security. So at the same time we must provide user selectable key-sizes, we also must provide different user-selectable digital signature algorithms. We must allow users to choose from different hash functions to build the key chain and make commitments, and also allow the use of other signature schemes with difference space/time trade-offs.

4. Miners selfishness and the unprotected Bitcoin user

From the game-theoretic point of view, Bitcoin miners do not have a strong incentive to protect end-users resources. In the long term miners may want to protect the end-users in order to maintain the value of their savings in the virtual coin, and the fixed cost of the infrastructure acquired for mining. But miners can at any time sell their coins and start mining for other P2P currencies, so the incentive is not strong enough. In the short term, they compete to collect fees, even if the transactions included in a block impose a high workload on the end-users. If miners were forced to store, transfer or compute much more data than end-users, then they would choose transactions of shorter length and lower CPU usage. In Bitcoin transactions are checked only once before the block mining process can start, and the quality and quantity of the transactions included in a block does not alter the winning probability significantly for a miner. The block size may affect slightly the dispersion time of a block across the network, and so may reduce the chances of a miner winning over a currently competing short-sized block. But currently this is not a limiting factor on miners, since blocks travel fast, and the diameter of the Bitcoin network is low. Also if a greater time is required to check a transaction, then the time when the block mining can effectively begin is postponed in that same amount. But transactions are checked only once, and each block mined requires a hashing effort orders of magnitude higher than the time required for transaction verification. So the CPU resources used in transaction verification during mining have little effect on the block cost and almost no effect on miners revenue. To create a free market in terms of security, we must build a system with the cheapest transaction cost at choice. Also, an end-user should pay the least possible cost to process other users transactions. In this paper we try to achieve these goals. Section 10.6 shows how to build a free market on security.

5. Redefining MAVE-3 payload for MAVEPAY

MAVE-3 is a digital signature protocol that allows massive realtime verification of signatures. The MAVE-3 digital signature protocol requires a semi-trusted third party called aggregator. Aggregators periodically publish blocks, and are similar to Bitcoin miners. A MAVE-3 signature consist of three commands (m1, m2 and m3) each broadcast to the network. In MAVEPAY, we have to wait for confirmation of each issued command, so the interaction to issue a MAVE-3 payment is:

- (1) Broadcasts m1 to the network
- (2) Waits for m1 to be included in a block.
- (3) Waits for some blocks to be mined for confirmation.
- (4) Broadcasts m2.
- (5) Waits for m2 to be included in a block.
- (6) Waits for some blocks to be mined for confirmation.
- (7) Broadcasts m3.
- (8) Waits for m3 to be included in a block.
- (9) Waits for some blocks to be mined for confirmation.

In a nutshell, message m1 serve as a commitment for m2, and m2 also serves as a commitment to m3, which carry a private one time key. The second message, m2, is the one that contains a payload, such as a signature and transaction specific information.

We redefine the payload field in MAVE-3 to include the following information:

```
\begin{split} payload = < S_A, f, R, [in\_subcmmd\_list][, out\_subcmmd\_list] > \\ in\_subcmmd\_list(i) = in\_subcmmd(i, 1), ..., in\_subcmmd(i, N_{IS}(i)) \\ out\_subcmmd\_list(j) = out\_subcmmd(j, 1), ..., out\_subcmmd(j, N_{OS}(j)) \end{split}
```

Where:

```
R: the receivers account (or output account). S_A: the amount of money to subtract from the input account S.
```

f: the fee to pay for the command.

 R_A : the amount to be transferred to account R ($R_A = S_A - f$).

We define a single *in_subcmmd*:

```
• CLOSE_CMD = \langle CLOSE \rangle.
```

We won't define any out_subcmmd yet.

A command is considered reliably issued if some confirmation blocks have passed after the command was included in the block and there is no competing alternate block chain.

Note that m1 and m2 should be included in a block even if they do not immediately pay fees to the block miner. The fee is only considered transferred after m3 is included in a block and confirmed. The fee can be sent to miner of the last command (m3) exclusively or split between the accounts of the 3 miners involved. In the later case, the accounts of the previous miners should not be closed before the payment is completed. To simplify the explanation, let assume the last miner receives the payment. Let F be the account number of the miner that mined the block containing m3.

In MAVEPAY each account is publicly identified by a N-byte binary number that is the MAVE-3 public key, where N is the size of the digest of a secure hash function (although it can be lower, if the digest is truncated).

Although MAVEPAY can be implemented to support scripts, in a manner similar to Bitcoin, in this paper we avoid using scripts to make the description clearer, and let us simply associate stored money with account numbers as traditional banking accounts do. It also helps to make payment messages shorter.

As in Bitcoin, the client application takes care of account management in a way that is transparent to the user, so the user can view his multiple accounts as a single account, with an unified balance.

MAVEPAY is build around the concept of a Transaction. A transaction is an operation on an account. A transaction that transfers money from one account to another is a Payment. In Bitcoin, each payment is a single command. In MAVEPAY, a payment consist of three messages broadcast to the network. Each of these messages is a Command. When a command is included in a mined block, we say it has been issued. Also a Payment may include the payment of a fee to one or more miners. Only the last command of a transaction actually executes it. The previous commands only serve to associate a (still unpublished) key to a particular payment so when the last command containing the key is issued there is no doubt which is the right payment to execute. Counterfeit commands, although they may contain a valid key, will not be accepted since a genuine and previously issued association will exists.

A difference between Bitcoin and the MAVEPAY scheme presented here is that we do not empty an account each time a transaction is issued from that account. We specify the amount of money to be transferred. Doing so we save the space required to specify a new account to hold the change.

One requirement for the scheme to be practical is that payments can be interleaved: it should be possible to start a new payment while the previous payment is being processed. Bitcoin already allows interleaving because payments require a single command and by specifying that the change is transferred again to the input account.

Because MAVEPAY payments span multiple blocks, we must explicitly design for interleaving. The argument to support interleaving is stronger in MAVEPAY since payments require longer confirmation periods than in Bitcoin. The reference implementation of MAVE/MAVEPAY allows a single payment per account per published block. It's easy to surpass this limit by modifying the protocol to allow multiple output addresses from a single payment. Then users can combine multiple payments in a single transaction, with the additional benefit of amortized cost.

It's important that the process of validating a payment can be carried out efficiently. Processing a payment means linking commands to payments and linking payments to accounts. To achieve this goal, the client application must maintain 3 data structures: $TABLE_1$, $TABLE_2$, and KEYRING as specified in MAVE-3. For MAVEPAY the last table will be renamed ACCOUNTS and will hold additional fields. ACCOUNTS has records $< S, x, LK_1 >$, that matches each account number S (MAVE-3 public key) with its amount x, where LK_1 is the last account key of the chain K_1 associated with the account. The data structure should allow efficient access by the key S.

To execute the payload, the procedure ${\tt EXECUTE_TRANSACTION}(m2,\!m3)$ is called.

Subprocedure ADD_COINS_OR_CREATE_ACCOUNT(R,A)

- (1) if the account S does not exists, add the record < S, A, S > to ACCOUNTS.
- (2) Otherwise, replace the record $\langle R, x, k' \rangle$ in ACCOUNTS with the record $\langle R, x + A, k' \rangle$.

Subprocedure CHECK_FUNDS(m2,m3)

- (1) Let $x = \langle S, a, LK_1 \rangle$ be the record found on table ACCOUNTS with key S.
- (2) If $a < S_A$ then abort.
- (3) Check the account is unblocked. If not, then abort.
- (4) Check that $R_A \leq S_A$. If not, then abort.

Subprocedure EXECUTE_TRANSACTION(m2,m3)

- (1) Call CHECK_FUNDS(m2,m3)
- (2) if $in_subcmmd_list$ contains the subcommand CLOSE_CMD, remove the record with key S from ACCOUNTS. Otherwise, replace the record $< S, x, LK_1 >$ in ACCOUNTS with the record $< S, x S_A, LK_1 >$.
- (3) call ADD_COINS_OR_CREATE_ACCOUNT (R, R_A)
- (4) If $f \neq 0$ call ADD_COINS_OR_CREATE_ACCOUNT(F, f).

To collect fees, each block can be signed by the aggregator with a classic digital signature algorithm or directly include the MAVEPAY address of the miners account.

To enrich the free market of security choices, it is also possible to create and let coexist MAVEPAY schemes of different key sizes and combine Bitcoin (or similar schemes) with MAVE-3 signatures, allowing payments to flow seamlessly between both type of accounts. Nevertheless, to achieve the benefits of a lightweight network, fees for transactions using Bitcoin should be much higher than fees required for MAVE-3. The distinction in fees allow users to switch between the schemes to make payments depending on the requirements for the transaction or the accounts involved.

As in Bitcoin, each user owns a number of accounts. In MAVEPAY, accounts have a predefined maximum number of payments they can issue. When this limit has been reached and a user wants to transfer money out of the account, the account must be closed afterwards and never used again. Although this may seem to be a restriction, there are multiple ways to overcome it:

- (1) The maximum number of payments can set to be as high as 1 million if you're willing to invest one second of processing in the account creation procedure.
- (2) A new account can be created to receive funds from a single sender in a single payment. This eliminates the risk that the account is being closed while some other payment is issued to that account. The 1:1 correspondence between payments and accounts is not a problem because account creation is a cheap and private operation in MAVEPAY. Also, most online shops already provide an unique payment addresses to each customer in order to distinguish the payer and provide grater anonymization for the transaction.
- (3) An account can be created to support multiple input payments up to a published time where it will no longer receive payments.
- (4) Persistent addresses can be created. In section 10.3, we'll discuss how to embed permanent user-friendly addresses or aliases in the block chain. A persistent address can be redirected to a new account once an account is closed.

(5) Keys can be recycled, associating a new key chain with the same account, as described in section 10.4.

6. Proof-of-work for commands

One of the key differences between Bitcoin and MAVEPAY is that Bitcoin has no "free rides". Every transaction can (and ussualy must) pay a fee. In MAVEPAY, we are required to create commands that cannot immediately pay fees. Then, the broadcast and inclusion in the block of such commands involves a risk for the network. An attacker may flood the network with dummy commands (spam) to consume all bandwidth or to fill the block with them, preventing legitimate users for making payments. To deter this attack, we suggest using a Proof-Of-Work (POW) for some commands. By requiring such proof of work, we verify that a user has invest at least a certain amount of computing time (e.g. one second) to create a valid payment. At least the first command of a transaction should carry a proof of work as an additional field (POW). Two alternative procedures are suggested for POW:

The preimage POW: The command is appended a random nonce (nonce) and the last block number seen (bn). The resulting message is called the packet, which is the message ready to be broadcast. From a packet, any client can compute the POW hash digest. The POW value is computed as a hash of the packet. The work requirement is that the digest is prefixed by some zero bits. The related attack is the partial preimage attack. This is similar to how Bitcoin makes POWs. The packet broadcast has the following properties:

- packet = < command, nonce, bn >
- tmp = Hash(command||bn)
- pow = Hash(tmp||nonce).
- pow is prefixed by some zero bits.

The Collision POW: The collision POW is based on the difficulty of finding partial preimage collisions of two messages. The *packet* broadcast has the following properties:

- $packet = < command, nonce_1, nonce_2, bn >$
- tmp = Hash(command||bn).
- $pow_1 = Hash(tmp||nonce_1)$.
- $pow_2 = Hash(tmp||nonce_2)$.
- pow_1 and pow_2 must be prefixed by some predefined number of zero bits and share some number of following bits (the remaining bits can differ).

The related attack is the partial collision attack. The advantage of the Collision POW is that it requires the benevolent user to do a collision search (or birthday attack) to efficiently create a POW with the less possible effort. Birthday attacks require large amounts of memory and so preclude the use of GPUs and other massively parallel computer architectures. An attacker is therefore less capable of using these technologies to mount a command flooding attack.

Note that packets are not entirely included in the block, only the command is. The validation and special treatment of the POW value in packets serves only to prevent command flooding before the command gets into a block. Clients must conform to the following protocol to ensure packets with less work are punished:

- (1) When the client application is installed it measures the number of work required for a POW that consumes one second, on average. If the preimage POW is chosen, then a number of leading zero bits L is selected. If the Collision POW is chosen, then a number of shared bits L is chosen.
- (2) When a node receives a command from the network with POW value with lower work than L, it is "slowed down" by re-transmitting it with low probability. On the contrary, commands with more work than L will be "accelerated" by being re-transmitted with higher probability.
- (3) Miners reject to include in blocks commands with a POW that represents a computation of less than a second in a standard computer.

If you want to let the big payment gateways (say PayPal) operate on a MAVE-PAY based network without too much additional load, then you can modify the requirement 3 so the miners can choose the required effort for a command to be included in a block. Then you can let the payment gateways arrange contracts with the top miners or mining pools to connect directly to them, and avoid using the network as intermediary transport.

7. Transaction Validation

In Bitcoin each transaction has to be validated at least twice to protect it from transaction flooding. First, when the transaction is broadcast across the network and secondly, when the transaction appears in a mined block. As MAVEPAY commands always carry a proof-of-work, we already have a measure to protect the network against command flooding. We suggest not to validate commands when they are broadcast by peers, and only check the proof-of-work of them.

8. Attacks specific to MAVEPAY

In addition to the possible attacks to MAVE-3, we describe attacks specific to MAVEPAY.

- 8.1. Command Flooding Attack. The command flooding attack on MAVEPAY is almost ineffective if commands carry a proof-of-work. Still, DoS of the MAVEPAY network can be mounted by a botnet. The attack requires thousands of machines to connect to the MAVEPAY network and broadcast association commands at once, multiplying the temporary storage requirements of benevolent clients. This attack cannot be easily distinguished from the normal operation. Still some practical protective measures can be taken:
 - (1) In the client application, new input connections are assigned lower priority than older ones. The number of new commands per second accepted in a connection depends on the connection priority. As connections mature, connection priority is risen.
 - (2) Each income connection is allowed to transmit almost the same amounts of commands of different types per second (associations and finalization commands). Statistics are maintained for each input connection to check for deviations. A high deviation from the expected rates is an indicator of flooding and so the connection is closed.

- 8.2. The isolation Attack. As in the delay attack, a user that is isolated from the network can be tricked into broadcasting the account key before the commands are issued in the real block chain. This attack requires that the attacker has the power to mine enough blocks in sequence to give the user the impression that his commands are being issued. If the attacker has 1% of the the network computing power, then he can build a chain 100 times slower than the network speed. If the payment confirmation time is 60 minutes, then the attacker must isolate the user for four days to allow the fake confirmation blocks to be received by the victim. This attack can be deterred by requiring that the clients detect any performance drop in the network hashing power, which is reflected in longer times between mining blocks. If such sharp downfall is detected, then the client should stop issuing payments until the situation normalizes. Other solution is that clients are always connected with some "friendly" nodes, and those connections are authenticated.
- 8.3. The 51% Attack. As in Bitcoin, a user that manages to acquire 51% of the hashing power of the network is able to disrupt the protocol. In Bitcoin, an attacker can revert other user payments or double-spend his money. On MAVEPAY the attacker is more powerful and can actually grab the money of any payment whose commands have been rolled back. This is clearly more severe. Also in MAVEPAY a command requires 3 times more confirmation blocks than in Bitcoin, for the same security threshold. The implementor may reduce the interval of blocks accordingly (say 1 block every 3 minutes, instead of 10) to compensate for the grater delay, but this change also reduces to cost of a sustained attack. If we assume the attacker has already build a super-computer to break Bitcoin or MAVEPAY, then both attacks are equally costly. But if we assume that the attacker will hire CPU/GPU time from some other source (e.g. Amazon Cloud Services) to acquire 51% of the network computing power, then the attack on MAVEPAY would be cheaper since less time will be required to replace the current block chain with an alternate chain (3 minutes instead of 10 minutes). We conclude that, for a strength against this attack similar to Bitcoin, MAVEPAY payments would take 3 times more to be confirmed. For additional protective measures designed to decrease the incentive for the attack, see section 10.1.
- 8.4. Eternal Storage of Transactions. One of the problems with Bitcoin is that, in order to prevent transactions of being broadcast ad-infinitum, each client maintains a hash table of cryptographic hashes of each transaction ever seen and avoids reprocessing a transaction previously processed by checking every transaction against this table. This protocol requires the eternal storage of transaction hashes. In MAVEPAY each command that is transmitted on the network is encapsulated in a packet. The packet specifies the last block number previously seen, and the POW field is applied afterwards. Each client must discard packets whose referenced block number is too old or too much ahead in the future. Whenever a client wants to notify its peer that a new packet has arrived, is advertises the packet hash along with the block number referenced in the packet. This notification procedure allows them decide whether or not they want to accept it.
- 8.5. Malicious Miner Attack. Miners tend to be benevolent to the network. They have invested time and money to build mining hardware to sustain a profitable mining business. Miner's earnings are received in the same virtual currency they mine, though fees and predefined prizes. They have no incentive to disrupt the

network, since a disruption is always followed by decrease in the exchange value of the virtual coin. Nevertheless, a malicious attacker may try to mine and create blocks that require too much bandwidth, computing power or storage to be widely and timely accepted by the network. A sustained DoS attack can create a currency crisis and a result in a run on the currency. To prevent the attack, the currency protocol must limit the resources that a block can request. Bitcoin protocol limits the number of signatures per transaction, and transactions per block. Since in MAVEPAY account creations are more costly than payments, the protocol must establish a limit on the number of account creations that a block can include.

9. Transaction Cancellation

As transactions are not atomic, it may be the case that a user wants to cancel the transaction before is has been finished. The network has already invest some effort in terms of broadcasting, checking and storing the transaction so cancellation should not be free. One solution is that, at the beginning, transactions pay a dynamically computed fee. The amount of the fee has to be calculated as an average of the fees paid in the last mined blocks weighted by the message size. The payment is taken from the input account and goes to the block miner of the first and second issued commands, but this payment is deferred by T blocks. Each account also holds a field blocked coins that specifies the amount of money blocked by fees of unfinished transactions. During the T-block time interval, whenever a client processes a new command of type 2 in a block, the blocked coins increases with the computed average fee. When checking for funds, blocked coins is subtracted from the account balance, so the spendable amount actually decreases. If the transaction is finished before T blocks have been mined, then blocked coins is updated and the fee specified in the transaction is processed as usual. If transaction of the type 1 and 2 commands (but not the type 3 command) have been issued and T blocks have passed, clients can proceed with the automatic payment. Type 1 commands are so small and require so little processing that they may be left exempted from fees. One problem with this type of cancellation is that an account holding less than the average fee will never be accepted for payment.

10. Improvements and Extensions

In this section we present improvements and extensions that can be added to the core protocol. To implement these extensions the payment checking procedures must be updated accordingly.

10.1. Multilevel keys and theft damage control. With enough monetary resources, an attacker may be able to create an alternate forged chain faster than the main chain. To maximize the profitability of the attack, the attacker would create blocks where all the payments done in the genuine chain are redirected to his own account. Even thought the cost of such attack increases with the length of the forged chain, the profit from the attack would be huge, since all the money from the accounts involved could be transferred. We must limit the profit from such attack. By using periodic permanent checkpoints, the maximum chain length that can be rolled back can be limited, but this limit may still not discourage the attacker. We cannot prevent the 51% attack, but we can minimze the losses it can cause by having different account keys, each one with a different maximum amount

of money per transaction, or maximum amount of money that can be transferred per time period. For example, with a single account key we may want to transfer \$10, and wait only a few confirmation blocks. If we do so, we're exposed to the risk of a 51% attack that rollsback the \$10 payment and creates a \$10K payment to one of the attacker's accounts using the key exposed. If we have two keys, one for payments up to \$10 and another for payments up to \$10K, we're affectively reducing the incentive for the attack.

We will extend MAVEPAY with accurate damage control: the configuration of multiple key chains, with increasing limiting amounts. The chains can be parallel or tree-like. If they are parallel, the account record size increases, if we use a tree-like chain (such as the K_2 chain), the command type 2 size increases by the same amount. A new $out_subcmmd$ command is defined to create an extended account.

Another use for multilevel keys is account reconfiguration. Suppose we want to allow each user to modify the behavior of the account such as adding another key chain or changing the maximum allowed amounts. Reconfiguration brings a high security risk, so it should have its own key chain.

```
CREATE_ACCOUNT_CMD =
    < CREATE_ACCOUNT, S, K<sub>C</sub>, limit_key_list >
limit_key_list =< limit_key(1), ..., limit_key(lk) >
limit_key(i) =< K(i), max(i) >
```

 K_C : is a key chain for account configuration. It allows to change the account behaviors.

K(i): is a new key chain whose transfers are limited by max(1) max(i): is the maximum amount of money that can be transferred with the key of the chain that ends with K(i).

10.2. Blocking payments to an account. A blocked account is forbidden to receive more payments. Because MAVEPAY accounts have a limited number of payments they can make, they must be emptied at the last payment. At that time the user may want to know exactly how much money the account is holding, without the risk that the account receives money meanwhile. By blocking an account the owner can check the balance. In the MAVEPAY payment protocol, when an account is closed it gets automatically blocked. We may want to manually block the account before we transfer money out of it. Another application of blocking is the creation of time limited accounts. A time limited account has the property that after a certain deadline, it is automatically blocked, without the need of interaction by the owner. Time limited accounts can be used to accept payments from many senders up to certain date.

To immediately block an account, we create a new command mb with format:

```
mb = \langle BLOCK, S, K_b \rangle
```

Where K_b is the following account key of an account S for a specific key chain in a multilevel key scheme, as described in the previous section. When a mb command is issued, every client blocks the account S to be the destination of payments.

To create time limited accounts, we create a new extra input subcommand that can appear in the field $in_subcmmd_list$, with format $< BLOCK, K_b, bn >$. Kb is the following account key and bn is the block number when the account should be blocked.

10.3. Friendly Persistent Addresses. One disadvantage of the protocol described so far is that addresses are not user friendly.

"2fd4e1c67a2d28fced849ee1bb76e7391b93eb12" is much hard to remember than "Jerry_Smith". Also it would be desirable that such friendly addresses could last forever, even if a certain account is closed. We can modify the protocol to allow friendly persistent addresses with little overhead. In fact, by using aliases, we improve the performance by reducing the size of messages. The idea is similar to NameCoin [3, 2], but tightly coupled with the main chain. One option is to create a new type of transaction that binds names to addresses but, since it won't be a payment, it cannot pay fees. To allow fees to be paid for binding aliases, we redefine a type 2 command so that it can specify either an account number (MAVE-3 public key) or an alias.

We define one new in subcmmd:

```
\begin{split} & \text{UPDATE\_ALIAS\_CMD} = \\ & < UPDATE\_ALIAS, opt\_new\_hash\_address > \end{split}
```

We define one new out subcmmd:

```
NEW\_ALIAS\_CMD = \langle NEW\_ALIAS, opt\_new\_friendly\_name \rangle
```

If the subcommand NEW_ALIAS_CMD is added to an $out_subcmmd$ field, then the new alias is created for the associated raw address R. If the subcommand UPDATE_ALIAS_CMD is added to an $in_subcmmd$ record, then the alias associated with the address S is updated to redirect to the raw address $opt_new_hash_address$. The embedding of bindings in commands of type 2 allows the alias to keep working after the processing of a command of type 3 that closes the account as long as the payments refer to the alias. We suggest using this extension with multilevel keys, since the unauthorized reassignment of aliases implies a high security risk for the user.

With a prefix to distinguish aliases from raw addresses, a payment can refer to any combination of them.

We'll show an example. Bob wants to send "Alice" the amount of money y. Bob creates a command with payload:

```
payload = \langle y + f, f, "Alice" \rangle
```

We must note that if we do not need to create a readable user friendly addresses, we can create arbitrary binary addresses. An alias as short as 8 bytes, allows 10^{19} unique addresses, that is more than enough.

10.4. Recycling an account. Accounts in MAVE have limited life, since keys in the chain can run out. Suppose Alice has an account S with alias "Alice". Suppose Alice continuously receives and send payments from her account through the alias, and the account is about to get closed because of lack of signing keys in the chain.

The first option is to create a new account Q, transfer the alias from S to Q, block the account S, check to see how much money is left in the account S, and transfer that money to the new account Q. A simpler option is to create a new $in_subcmmd$ that replaces the current last key in the chain with a new key that was build from a new key chain.

UPDATE_KEY_CMD =
$$< UPDATE_KEY, k_c, new_lk_1 >$$

When the payment containing this subcommand is processed, the field Lk_1 from the table ACCOUNTS is replaced by new_lk_1 . The subcommand must include the next valid key k_c from the configuration key chain as defined in 10.1.

Still another option is to create a new identifier REMAINING that, when given as S_A field, specifies that the remaining money in the account must be transferred. Now, Alice wants to close the account A, and transfer all funds to a new account Q and preserve the alias. She creates the command with payload:

$$payload = \langle REMAINING, f, \langle UPDATE \ ALIAS, Q \rangle \rangle$$

With any of these options, the alias "Alice" works effectively a virtual account number that is always open.

10.5. Balance sheets. The idea of balance sheets (or checkpoints or snapshots) was suggested to improve Bitcoin storage and reduce the initial delay for new clients [8]. In Bitcoin each account record would require 80 bytes to store the address plus scriptSig. Since each full Bitcoin transaction requires on average 459 bytes, and assuming each transaction creates a new address, the space required for a balance sheet would be 17% of the block chain size. In MAVEPAY, the size of a account record can be as short as 20 bytes, then a MAVEPAY balance size is only 25% of a Bitcoin balance.

One advantage of using balance sheets is that whole balance database can live in RAM while the client application is online. The storage of the balance in RAM can speed up account lookups and payments by at least a factor of 1000. But to take full advantage of balance sheets we must also punish the use of multiple accounts in favor of a few accounts per user (or a single account, at best). To encourage the use of few accounts per user, we can:

- (1) Create an specific subcommand to open a new account.
- (2) Force the creation of accounts to pay a high fee, or a periodic fee to keep the account open.
- (3) Force accounts created by payments to be temporary.

10.6. MAVEPAY / Bitcoin Hybrid and fee rates. In this section we discuss the possibility to create an hybrid system MAVEPAY / Bitcoin. As both schemes rely on miners, we can let them create blocks mixing Bitcoin transactions and MAVE commands. Bitcoin transactions can be distinguished from MAVE commands by special prefixes. Output addresses in MAVE-3 can be extended to specify a Bitcoin scriptSig. Bitcoin output scripts can be extended to specify a MAVE addresses. But to take advantage of the performance of MAVE and the flexibility of Bitcoin we must encourage users to select the appropriate type of account. Accounts that requires maximum security (such as banks) could use Bitcoin accounts, while all other users would use MAVEPAY accounts. One way to encourage the right selection is to adjust transaction fees according to the real cost of verifying them, creating fee rates. Considering current technology and bandwidth of an average user, we would need to give an incentive to use MAVEPAY accounts in favor of Bitcoin. This could change if one day computers come with specialized hardware to accelerate some cryptographic algorithms. We propose three different schemes that can be used to provide fee rates: fee confiscation, command difficulty and limited size blocks.

10.6.1. Fee confiscation. In this scheme, part of the fees collected by a miner get "confiscated". When a command/transaction with fee f is included in a block, the miner applies a predefined multiplier x to the fee f. The miner can only collect x * f and the rest is confiscated. The multiplier x is always lower or equal to 1. The longer the message, the lower the multiplier. The slower the cryptographic operations required by the transaction, the lower the multiplier.

As an example, if CPU usage was the only factor to consider to calculate the cost of a transaction to the network, then a transaction which requires 100 times more time to evaluate than other would have a multiplier that is 100 times lower. Because miners always choose the transactions that give them the higher reward, then users would be forced to compensate the punishment of confiscation by increasing the fees by the same factor for those commands.

CPU usage is not the only factor to consider when calculating the cost of a transaction to the network as a whole. All expensive resources already described must be considered to design a realistic function that takes into account average costs and tries to anticipate how those costs will evolve in the future.

As fees are reduced by the multipliers, is necessary to restore the remaining money (1-x)*f to the network to avoid destroying it. One possible solution is to accumulate all the remaining fees and setup a price to be awarded to the miner of the following block. To prevent the miner from trying to delay broadcasting a block in order to mine the next, we can setup the price to be awarded to the miner of some blocks ahead (e.g. ten blocks).

This automatic prize generation may give an incentive for the casual miner not to include so many transactions, since a fixed reward (higher than the transaction fees) may exist. But since including transactions in a block requires very little resources, here is no reason not to include all known transaction and collect all possible fees. For the miners who have a high percentage of network computing power (like mining pools) obviously no such incentive exists, since including less transaction imply being awarded less money as prices in following mined blocks.

If a payment is sent to an account that was not previously created, a higher fee must be paid for the additional storage required. One alternative is that accounts created by payments (not explicitly pre-created by CREATE_ACCOUNT_CMD) are temporary (e.g. money is kept only for a month). Before the period finishes, the owner must transfer the money to a pre-created account or pay a special fee for the creation, otherwise the account contents is returned to the network as a miner's prize.

10.6.2. Command difficulty. We can force the miner to actually work more depending on the commands included in the block. Each command, depending on the type and length, would add difficulty to the target difficulty of the block. The ratio between difficulties of different commands would be set by the network designers to match the actual verification costs to the network.

10.6.3. Limited size blocks. If block size is limited by design and the block chain is generally "saturated", then miners have an incentive to choose the smaller commands that give them higher fees. Also it is possible to decrease the maximum number of allowable commands per block depending on the type of the commands included. The problem with this scheme is that blocks sizes vary and it's difficult

to artificially force miners to leave commands out of blocks, even by dynamically setting the maximum block size.

10.7. Payer identification. One of the benefits of a 1:1 relation between payments and accounts is that, in case of a user that simultaneously receives multiple payments, payers are identified by their destination account. In Bitcoin, the script has payload to include payer identification strings. In MAVE we can also extend the payload of a message of type 2 to include a payer identification string. The string, along with the transaction message will be cached by the network until the next balance sheet is ready.

11. RESOURCE USAGE

In MAVEPAY each payment requires 3 commands. We will check the space requirements for the cheapest MAVEPAY scheme that is still secure for every day use as of 2012.

Assumptions:

- (1) For account keys, a truncated HASH with 80 bits digest is used (N=10).
- (2) An alias system with an average alias of 8 bytes (64 bit address space)
- (3) A truncated hash commitment with M=10 (80 bits of pre-image security)
- (4) Amounts are expressed as a compressed unsigned int, using special prefixes for extending the field size.
- (5) Average amounts requires 4 byte unsigned int.
- (6) Each identifier consumes a single byte
- (7) Field separators consume a single byte, and are included only for non-self delimiting fields.

Then the following table summarizes the average sizes of commands for MAVE-PAY:

- Type 1: 10+1=11 bytes (1 identifier, 1 commitment)
- Type 2: 8+20+20+10+1=63 bytes (2 amounts, 2 aliases, 2 account keys, 1 commitment, 1 identifier)
- Type 3: 1+10=11 bytes (1 identifier, 1 account key)

The average payment size for MAVEPAY (omitting other headers such as version information) is therefore 11+63+11=85 bytes. This is less than 19% of the average Bitcoin payment size, which is 459 bytes. In other words, MAVEPAY can support almost 6 times more transactions for the same bandwidth and storage space. This figure does not take into account possible optimizations to the Bitcoin protocol to compress each transaction.

In MAVEPAY each payment requires the computation of 3 hash digests. In a modern computer the time required to compute a SHA-1 digest from a 40 byte binary string is approximately 2 uS, so 3 digests takes 6 uS. Verifying a ECDSA signature takes on the same computer 8 ms (more than a thousand times more). The MAVEPAY bottleneck is not hashing, but accessing and updating the tables (TABLE_1, TABLE_2 and ACCOUNTS). MAVEPAY requires approximately 10 look-ups per payment, instead of 3 look-ups required by Bitcoin. But TABLE_1 and TABLE_2 should be relatively small, so these accesses should not be taken into account.

Now we'll estimate the maximum number of transactions per second MAVEPAY withstand assuming an average home computer.

We explore only one setting for the client: balance-in-RAM. In this setting, the whole account balance for the network is stored in RAM while the client is online. In balance-on-HDD setting, the account balance is stored in a hard drive. This distinction is appropriate since the time required to process a payment is much lower when we can assume the whole network balance is stored in RAM.

11.1. The cost of 1K payments/second. To achieve 1K payments/second we must make use of the all the described enhancements in section 10. To compute the cost of a manage a MAVEPAY node we have assumed:

Typical US monthly residential rate: $0.11~\mathrm{USD}~\mathrm{per}~\mathrm{KW/h}$

Average Computer Power consumption: 100 Watts

Desktop personal computer cost: 1000 USD

PC amortization time: 4.5 years Available bandwidth: 1 Mbps

Average monthly cost of Internet in US at 1Mbps: 3.3 USD

Maximum MAVEPAY CPU usage: 15% Maximum MAVEPAY bandwidth usage: 50% Degradation/Failure due to MAVEPAY use: 50%

Monthly amortization cost due to MAVEPAY use: 9.26 USD

Average alias size: 10 bytes

Average account key size: 12 bytes (weighted average of key sizes of low risk (10 bytes) and high risk (22 bytes) accounts at a 5:1 ratio)

Setting: Balance-on-RAM

Initial accounts in existence: 100,000

Accounts created a year: 5 M

Payments per second: 1000 payments/second

Taking into account these assumptions, the cost to run a MAVEPAY node is $13.17~\mathrm{USD/month}$.

It it difficult to estimate the initial number of accounts and the number of accounts each user will create, since it depends on the fee rates established for account creation and the need of anonymity. During 2011, Bitcoin had between 30k-60k active users [10]. The total number of casual users during 2011 (while some may be inactive) is approximately 740K. If each active user were the owner of a single account, then the whole balance sheet would be no longer than 3Mb. The balance sheet for ten million users fits into 500 Mb. For comparison, the current cost of Bitcoin at only 10 payments/second is 12.28 USD/month. Achieving 1000 payments/second in Bitcoin by stacking home PCs would probably cost over 500 USD/month (this amount is lower if the user can optimize power consumption as he builds a small data center or using specialized hardware to speed up ECDSA operations).

11.2. **Anonymity vs Scalability.** In MAVEPAY, anonymity is punished in favor of scalability. A user that wants to stay completely anonymous requires transactions that specify both input and output addresses in raw form. Such users will not use the alias system. Payments that specify raw addresses tend to be longer. Also each payment would need to use money from multiple input accounts, using

consolidation, since a different account would be used for each payment received. Therefore such payments would pay higher fees.

11.3. Beyond 1K payments/second. Moore's law predict the doubling of transistor density every two years, so both RAM size and CPU power increases accordingly. Kryder's Law predicts magnetic disk areal storage density doubles approximately every 18 months. Residential link bandwidth doubles every 18 months. Also, desktop computers are replaced every 4.5 years on average. This facts imply that every time the user replaces his computer every 4.5 years, he has at least multiplied by 4 the main computer resources required by MAVEPAY. Since we assume the number of MAVEPAY accounts will increase steadily, the balance database increases over time. We can safely assume a 2x increase in accounts created (the user base) every 4 years. The payments processing capacity can safely double every 18 months, since is mainly restricted by available bandwidth.

12. Conclusion

We have presented MAVEPAY, a new class of payment scheme that, like Bitcoin, relies on the concept of the proof-of-work block chain. We have shown how MAVEPAY design differs from Bitcoin, and analyzed pros and cons of each design. MAVEPAY design allows much better scalability and greater performance, since it relies on faster cryptographic constructions. Also we have shown that MAVEPAY require greater care for confirmation blocks, and is exposed to greater danger from block chain rollback attacks. We have shown how to create a free market where accounts holding higher amounts of money pay higher fees to achieve higher security threshold (the Least Required Security), can reduce the fixed costs of operation for the average clients. We have described enhancements (section 10), such as account creation fees, periodic "balance sheet" cleanups and persistent aliases, that allows us to create a truly lightweight peer to peer currency network that process up to 700 payments per second, and 5 million new users a year, with current home computer technology, and an average Internet connection.

We summarize the properties of MAVEPAY:

- (1) Interleaved payments
- (2) Cancellable payments
- (3) User-friendly account aliases
- (4) Protection against the Delay Attack
- (5) Protection against the $O(c^2)$ Attack
- (6) Protection against Command Flooding Attack
- (7) Protection against the Isolation Attack
- (8) Protection against the 51% Attack
- (9) Protection against Eternal Storage of transactions

The first 5 properties are given by MAVE-3. The last properties were discussed in section 8.

REFERENCES

- [1] ? Bitcoin scalability. Apr. 2012. URL: https://en.bitcoin.it/wiki/Scalability.
- [2] ? Dot-BIT project. Jan. 2012. URL: http://dot-bit.org/Main_Page.
- [3] ? Namecoin. Jan. 2012. URL: https://en.bitcoin.it/wiki/Namecoin.

- [4] VAMPIRE Virtual Applications and Implementations Research Lab. eBACS: ECRYPT Benchmarking of Cryptographic Systems. Apr. 2012. URL: http://bench.cr.yp.to.
- [5] Ark. Solving the symmetric key distribution problem for digital signatures using the BitCoin peer-to-peer infrastructure. July 2011. URL: http://archimerged.wordpress.com/2011/07/14/solving-the-symmetric-key-distribution-problem-for-digital-signatures-using-the-bitcoin-peer-to-peer-infrastructure/.
- [6] Sergio Demian Lerner. MAVE: new lightweight digital signature protocols for massive verifications. Apr. 2012. URL: http://bitslog.files.wordpress.com/2012/04/mave.pdf.
- [7] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: (). URL: http://fastbull.dl.sourceforge.net/project/bitcoin/Design\%20Paper/bitcoin.pdf/bitcoin.pdf.
- [8] Piuk. A proposal for a scalable blockchain. Nov. 2011. URL: https://bitcointalk.org/index.php?topic=52859.0.
- [9] R. L. Rivest and A. Shamir. "PayWord and MicroMint: Two Simple Micropayment Schemes". In: *Proc. 4th International Security Protocols Conference*. 1996, pp. 69–87.
- [10] RowIT. RowIT Bitcoin Peer to Peer Network Status. Jan. 2012. URL: http://bitcoinstatus.rowit.co.uk/.
- [11] The Bitcoin wiki. The Bitcoin wiki. Apr. 2012. URL: https://bitcoin.it.

 $E ext{-}mail\ address: sergiolerner@certimix.com, sergiolerner@pentatek.com}$