Democratic Cryptocurrency Unity

Cryptocurrencies came to the world in the recent decade and attempted to put a new order where the financial system is not governed by a centralized entity, and where you have a complete control over your account without the need to trust strangers (governments and banks above all). In this paper an innovative approach for cryptocurrency is introduced. This approach is based on what we are already used to - banking and democracy. Our banks, just like normal banks, keep their clients' money, and perform their clients' requests. However, your bank cannot do anything in your account without your permission, because of the cryptographic scheme, and its entire operation is transparent so you don't have to trust it blindly. The democracy means that every operation performed by the banks (e.g., committing a client transaction) is accepted by majority of the coin holders, in a similar manner to representative democracy, where the banks are the representatives, and where each client implicitly delegates his voting power (the sum of money in his account) to its bank. A client can switch banks at any moment, by simply applying a corresponding request to the new bank of his choice (with the cost of paying commission). The presented approach employs the advantages of centralization while still providing a complete trustless and decentralized system. By employing concepts from everyday life and attaining high throughput and low latency for committing transactions, the hope is that this paper will put the foundations for a cryptocurrency that can be truly used as our daily basis coin.

1. Introduction

Cryptocurrency became globally known in recent years, especially by the emerge of Bitcoin. Although it is the main first decentralized coin, and despite its glorious success (for the moment), Bitcoin has many known. Among its flaws are the time it takes a transaction to be accepted, its limited transactions throughput, and finally, the great energy consumption involved in keeping it alive.

There are numerous works and other cryptocoins that are trying to fix these flaws. Especially, there are strong environmental and economical reasons to alleviate its energy consumption that follows from its innovative mechanism for solving the "Consensus in the Permissionless Model" problem. This problem is one of the main challenges that cryptocurrencies deal with, and it originates from the distributed nature of the cryptocurrency. In real life, you have either physical money, e.g. metal coins and bills, or otherwise your bank keeps record of the money you deposit. In cryptocurrency however, you have no physical money, and there is no single entity that can be trusted to tell you how much money you (or someone else) have. Instead, the amount of money you have should be in consensus between the other users of the cryptocurrency – everyone should agree, somehow, that you have that specific amount of money. Moreover, this vague consensus should be on all the transactions (money transfers) that are committed. This is required in order to prevent double spending, where one can pay the same coin more than once, as each payee doesn't know that the money was also payed to others. Once there is a

consensus on one of that user's payments, the corresponding payee can be sure that he got the money (because everyone agree that he got the money). A second payment with the same coin will not be accepted, as the corresponding (second) payee will see that it is not in the consensus. The big question is how can you reach such consensus in a settings where you can trust no one, and where it is not generally defined who the other players are (i.e., a "permissionless model").

The common denominator for probably all the numerous different cryptocoins is that there are roughly two groups - users and administrators. The users are the simple persons/clients that want to hold coins and use them for any type of trade or investment. The administrators role is to make sure that there is a consensus concerning the current balance of the user accounts and concerning the committed user transactions. The administrators must invest resources such as computation, storage and bandwidth in order to manage the consensus. They usually have a simple incentive to do so - they receive coins for their work (either by commission from transactions, or by creating money from thin air, i.e., stamping new money). The coin they receive plays a double role - it provides both revenue for their investment, and an incentive to keep the stability of the coin, as otherwise the coin value will decrease and the administrators' true gain will decrease as well (the coins they will receive shall have lower value). The relationship between the users and administrators is usually not well defined. It is only required from the administrators to form some kind of a network (so they can speak with each other in order to reach consensus) and from the users to be able to transmit their requests to at least one of the administrators. In the approach presented in this paper there is a straightforward relationship between the users and the administrators.

While the administrators are those that manage the consensus, we claim that the consensus should be between the users. Finally, it is the users who should be interested in the consensus. Without consensus there is no value to the coins they hold (as there is no agreement on the amount of coins each user holds). The more coins you hold, the more responsibility you have for its future (as you will lose more if its value will drop). The way we can coordinate between all the different users is by means of democracy - the majority of the "people" determines. Note, however, that the "people" in our case are not exactly the users, bur rather the coins. As coins belong to users, each user has a voting power that is proportional to his amount of coins. The problem is that the users are too many, and asking all of them to vote on each decision (i.e., transaction) is not practical. Recall that in a democracy the people don't need to accept every rule, they just need to choose their representatives to make the decisions for them (or on their behalf). Thus, our users simply need to choose representatives. The most trivial representatives are the administrators, whose job is to keep the consensus. We shall now talk about the identities of the administrators.

In the real world we don't keep the money ourselves (at least most of the money). Instead, we let someone (the bank) save it for us. When we want to use this money we simply address our bank (either directly or indirectly). Life was good and simple, and then came the cryptocurrency fellows and said "whoa, you need no banks! Your money can be in your hands! (by the agreement of the rest)". "Very good!", you smile, "No banks means no commission!". "Oh well... recall that cryptocurrencies need to achieve agreement on the accepted transactions, and for that cause there must be those administrators that put effort in it (computation power/storage/bandwidth), and... no much altruism these days, so they

must get some money for their efforts". While real banks physically keep our money, the administrators only have to manage the information about how much money each user has. In today world, where most of our money is anyway just numbers (and not something physical), is there a really difference between the two? Well, there is one important difference - the banks are centralized entities, whom we must trust, while the administrators are not. We don't need to trust the administrators because their way of action is transparent, and in fact each of us can become such administrator (or at least can gather the information that they get) and check for himself the correctness of the consensus.

While the common cryptocurrency scheme has moved away from banks, we return to them. Our banks, just like administrators in other cryptocoins, will be responsible to manage the consensus. And just like other administrators, their way of action will be completely transparent so they don't have to be trustful. From the user side, this scheme is just as in real life: The money of each user will be virtually deposited in a bank (an administrator) of his choice, and for every operation that the user wants to perform he should send his request directly to his bank. Of course the user must also have the option to switch banks, so he won't lose his money in case his bank fails or otherwise ignores his requests. The user can do so by sending a "draw my account" request to another bank. This scheme introduces several advantages that follow from the centralized nature of banks. From the user side, it means the user has a single and known address for all of his issues. From the administrators (the banks) side, it means simple and defined power distribution (each bank has the power according to the total amount of money of its clients). Another important advantage follows from our consensus mechanism, that will be now described shortly.

Let's assume I ask my bank to transfer money from my account to another. As we are living in a democracy, the transfer must be accepted by the holders of the majority of the coin (i.e., a group of coin holders that together posses the majority of the money must accept this transfer). In our approach each bank represents its clients and `votes' on their behalf. Roughly speaking, each bank gets its voting power according to the sum of money in the user accounts it manages. Once the banks agree on a transaction, we can see it as if the money holders themselves agreed on that transaction (as each money holder granted his voting power to his bank). So far we have only described the basic principle of our consensus - a set of banks that represent majority of the coin holders must agree on each transaction. The question is how do we implement it? In other coins there is usually a single global ledger that lists all the transactions, and everyone (should) agree on this ledger. By observing the agreed ledger, everyone can tell, out of two (accepted) transactions, which one comes before the other. I.e., there is an agreement on the total order of the accepted transactions. This ledger is the famous "blockchain" - a chain of blocks, where each block contains a set of transactions. However, such agreement on the total order of the transactions is superfluous and a source of many problems, as it brings together all the administrators into a single condensed point - appending a new block to the blockchain. The obvious question is who will be the one that defines the next block? The solution of Bitcoin (for example) to this question causes a grave waste of resources, with big latency and small throughput for committing transactions. Other solutions might mitigate some of these problems, but no solution can be optimal in the sense that such global blockchain is more than what is truly required.

Instead of using a global ledger (i.e., a blockchain), in our approach we let each bank to manage its own ledger - a separate ledger(/blockchain) for each bank. The transactions that appear in such "private blockchain" of a given bank are only transactions that are committed by the clients of the same bank. I.e., it lists only transactions that transfer money from that bank's clients to other clients (other clients either of the same bank or of other banks). If we want to compute the balance of a client, observing only the blockchain of his bank is not enough, as it doesn't list money that he receives from clients of other banks. The information on such money transfers is found in the blockchains of the other banks (the banks whose clients transferred that money). As banks must compute such account balances to make sure the transactions are legal, they must hold all of the existing blockchains. Moreover, each block in a blockchain will contain pointers to blocks in the other blockchains (to the most recent blocks, at the time of creating that block). More information appears in Section 3.

So, when I send a transaction to my bank, my bank will check this transaction to make sure that I have the required money to spend, then it will probably group it with parallel transactions of other users into a block, and it will chain this block to its blockchain. Next, my bank will send this new block to all the other banks, in order to receive their acceptance for the block. Once a majority of banks (by means of voting power) sent their acceptance for the validity of the block, my bank has the proof that the block is valid. More information concerning the exact protocol appears in section 4.

Another important aspect in the presented approach is the incentives for the banks. As you might guess, the banks will live on your commissions. More precisely, you will pay a commission on each of your issued transactions. The commission you will pay on such transaction will be divided between your bank (probably an half of it) and the banks that accepted this transaction (the other half, divided between those other banks based on their relative voting power). The commission amount will be a percentage of the transaction size. The exact percentage will be proportionate to the voting power of the accepting banks, in order to encourage full participation in the agreement process. I.e., the more banks agree on the transaction (or, more precisely, on the block that contains this transaction), the bigger the commission is. As the issuing bank receives a fixed share of the commission (and possibly a bit extra), the commission it receives will be bigger as more banks agree on the transaction. This incentivizes the issuing bank to ask for the acceptance of as many banks as it can. The other banks are also incentivized, as each bank that participates in the agreement process receives some part of the share.

Note that real banks must have some capital of their own (by regulatory requirements, in order to assure their stability). The original reason (financial stability) is not true for our banks, as our banks cannot use their clients' money, in opposed to real world banks. Yet, this is a good idea to force our banks to also have some capital in a private account of their own. In our case this is used as an incentive to make sure the banks keep the protocol. A bank that does illegal stuff, will simply lose its money. More precisely, if a bank does something maliciously, and the other banks have the proof for that, then the other banks will ignore its future issued transactions, so it won't be able to spend the money in its private account (and this bank will also lose its voting power). That bank's clients, however, won't be affected as they can simply address other banks and ask them to "draw" their accounts. We incentivize the banks to put money in their private account by defining the bank voting power (in the consensus and commission distribution processes) to be a

function of both the sum of money that belong to its clients (as been said before), and the money in its private account. In particular, this function is defined such that a bank with empty private account will have no voting power at all. The more clients the bank shall have, the more capital it must hold, if it wants to use its full voting power (recall that more voting power means bigger share in the commission).

In the common cryptocurrency scheme, each user chooses a pair of private and public keys. The private key is used by the user for encoding digital signatures, and the public key is used for verifying the user's digital signatures and as the account number. In our approach however, we want the user's account number to be related to a specific bank of his choice. For this cause, each bank has to chose a pair of private and public keys (just like the simple users), and a user's account number is in fact a combination of his bank's public key and his own public key. The bank's public key can be seen as the "branch number", while the user's public key is the "inner account number" in that specific bank.

Let us summarize our new cryptocurrency scheme: First, when I "open" a new account, I do it using my ordinary private and public keys (just as in other cryptocoins). But, I also choose a bank and using its public key as my "branch number", so that my full account number comprises of "branch number" and "account number", just as in real life. When I want to issue a transaction, I send it to my bank. My bank is in charge to make sure other banks acknowledge this transaction, i.e., that the representatives (the banks) of the majority of the money holders accept it. Of course I shall pay commission on that transaction (that will be divided between the banks that took part in the process). And finally, if I don't like my bank, then I can address another bank and ask it to "draw" my account into it.

The presented approach has the following advantages:

-Agreement mechanism without superfluous energy consumption.
-Completely decentralized and trustless system.
-Just as in real life, you address all of your requests directly to your bank.
-As each bank updates its own ledger, we can have higher transaction throughput with lower latency. As a result of the lower latency, this coin can be used as an everyday payment method.
-The banks have incentives to cooperate with each other and to comply with the protocol.
-The banks have also incentives to give their clients an appropriate treatment, or otherwise the clients will move to other banks (and less clients means lower commissions).

2.Related Work

Maybe the greatest challenge of cryptocurrency is the consensus on the money balance and/or committed transactions. The most prominent approach to this problem, introduced by Bitcoin, is the famous blockchain. The difficult question is how a new block is added to a given chain, in a way that everyone will agree that this block is indeed the next block in the chain. The first approach, Bitcoin's approach, is by a race based on computational power. The chance one has to win such race is roughly equal to its computation power divided by the overall computation power that participate in this race. By assuming that most of the computational power is in "good" hands, that will recognize the true winner of a race, it is argued that it will benefit everyone to recognize that winner as well, thus, reaching

consensus. This concept is called Proof-of-Work (PoW). The great downside of PoW is its resource consumption.

In order to alleviate the resource consumption, the idea of Proof-of-Stake (PoS) was born, where instead of deciding the next block by stochastic means, based on ownership of external resources (such as computation power), we can decide in a deterministic fashion based on inherent resources - the coin itself. The idea is that if you have more of the coin, you can get the chances to create more blocks. While this solves the energy consumption problem, it doesn't truly solve the other downsides of Bitcoin - high latency and low throughput for committed transactions.

Achieving high throughput and low latency is problematic when relying on a single main blockchain that everyone must agree upon. In order to solve this problem, one option is to do stuff outside the main chain (e.g. [10]), and update the main chain only when necessary. Another option, however, is to completely avoid a main blockchain (e.g. [11,7]). Note that the blockchain is in fact an implementation of state machine replication (SMR). For the cryptocurrency scheme, an SMR is rather "overkill", because a total order of all the committed transactions is not truly necessary. Rather, the true importance is on the order of the transactions that are committed by each account separately. I.e., if a specific user committed two transactions, we should know (and agree) which one comes first, so in case that user doesn't really have enough money for both of the transactions, we shall all agree which of the two will be committed (the one that we all agreed to be the first of the two). An approach taken by Nano, for example, is to use a separate blockchain per account. More precisely, each account orders its own outgoing and incoming transactions in a single list. Note that an outgoing transaction in the "blockchain" of one account will appear as an incoming transaction in the "blockchain" of the credited account. Thus, the different private blockchains are all connected. By having the set of those `private blockchains' you can make sure that they are all correct, i.e., that each outgoing transaction has the money to spend. As each user maintains his own local blockchain, the Nano coin can be treated as "cash" - the money is in your hands. However, there is still the need for consensus on that cash you holds (and on every transaction you add). In Nano the administrators are nodes that accept the users blockchains, and solve conflicts when they appear. Maybe the greatest downside of Nano is that those nodes (the administrators) get no revenue for their work, which means they have no incentives to invest resources. Our solution is similar to Nano in this sense, as we also use several parallel blockchains instead of a single blockchain. However, instead of the fine granularity of the `blockchain per account' (where each user must maintain its own personal blockchain), we present a coarser granularity, `blockchain per bank' (where each bank maintains a blockchain that contains its clients transactions).

Concerning the consensus mechanism, we employ `democracy' where a single coin means a single vote and transactions are accepted by a majority. The voting power of each coin is in fact delegated to the bank in which it is deposited. The idea of delegating voting power to an administrator seems to be, unfortunately for the author, not new, see e.g. [7,1,2]. The advantages of delegating the voting power is that we can apply consensus by `democracy', where the coin holders are effectively those that decide. Maybe the most prominent problem in delegating power is the `indifferent user. The problem is when users don't really care to whom they delegate their power, or when the users aren't forced to delegate their power at all. The result might be that most of the honest users delegate their power to a non functioning or malicious administrators, or even don't delegate their power

at all (in which case the rest of the users, that many of them might have malicious meanings, virtually hold more power than their true share).

In our approach the users cannot truly be indifferent. The reason is the close connection between the users and the administrators, or more precisely the `intimate' connection each user has with one specific administrator - his bank. As our users must choose a bank, and as they can get service only from that bank, they cannot delegate their voting power to a bank that is not functioning or that was proved to be malicious (a bank that acts maliciously will be ignored by the rest of the banks, so it won't be able to give services to its users).

While some of the concepts, such as managing separate blockchains and delegating the voting power, appear also in earlier works (and particularly they both appear in Nano), the way they are integrated in this work is innovative and provides many advantages over other solutions.

3.Settings

There are two types of accounts: user accounts, and bank accounts. A user account is the combination of the user's public key and a bank's public key, with the user's private key used for creating digital signatures. A bank account (also defined as the bank number) is simply the bank's public key, with its private key used to create signatures. A simple transaction is a coin transfer request, of some amount, from one account to a second, digitally signed by the first account's private key. A transaction must have another unique info (such as transaction number, hash of previous transaction, current date and time or any other nonsense) in order to avoid reuse of the transaction.

Note that a user should send his transaction to his own bank. His bank will make sure that this user has enough money, and then it will put this transaction in a transaction block, possibly together with other transactions that this bank has received from its clients. All the transaction blocks of a given bank will be chained in a list we call blocklist (i.e., a blockchain). Observing the set of blocklists of all the banks, we can compute the total balance of the existing accounts by applying all the transactions that appear in these blocklists. To such set of blocklists we call a history. More precisely, a history H is a sparse tuple where each cell in the tuple corresponds to a bank number and holds that bank's blocklist. Following the previous talk, by applying all the transactions that appear in a history, we get the total balance of the active accounts. That total balance is in fact a snapshot of the system.

Now consider the case where a bank wants to issue a new block, containing new client transactions. The bank must first be sure that the relevant accounts have the money to spend. For that cause it must observe the current history (that it is aware of), and compute the total balance. After the bank made sure that all the transactions are legal (i.e., the relevant accounts have the money to spend), it should send its new issued block to the other banks. The other banks clearly don't trust that bank, and they shall check for themselves that the transactions in that block are legal. I.e., they should compute the current balances themselves, and make sure the relevant accounts have the money to spend. To perform this check correctly, the computation of these banks must yield the same balances as the computation of the issuing bank (i.e., they should have an identical snapshot of the system). This will be true if they will see the same history as the issuing bank. For

that cause, the new block must contain extra information that will instruct the other banks how to construct the correct history. More precisely, each block includes pointers to the last blocks of the other blocklists (see Figures 1 and 2). To save data size, if there are no changes to a given blocklist since the previous block, that came before the newly issued block, then the new block can avoid including the same pointer. This way, the block contains only "updates".

More formally, every transaction block contains the following:

- The issuing bank number and the block serial number.
- Hash of the previous block in the list, or hash of another special block (the genesis block - see below) if it is the first block in the list.
- A set of transactions that transfer money from the bank clients.
- A (possibly empty) set of references to blocks from blocklists of other banks that appear in H. Each reference is actually a triplet of bank number, block serial number and hash of the specific block. There is at most one reference per bank.

Figure 1 shows a (simplified) block structure. Figure 2 shows a history that contains the block depicted in Fig. 1 (with blue edges), with pointers toward the last blocks that were known when issuing this block. When checking the correctness of (b_1; #2) in this example, we should compute the balance after applying the transactions from (b_1; #1), (b_2; #1), (b_3; #1), (b_3; #2) and (b_3; #3).

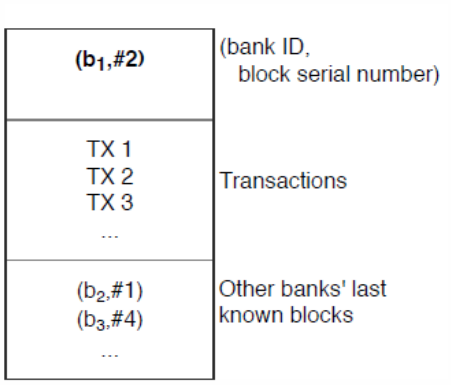

Fig. 1. Transaction block structure.

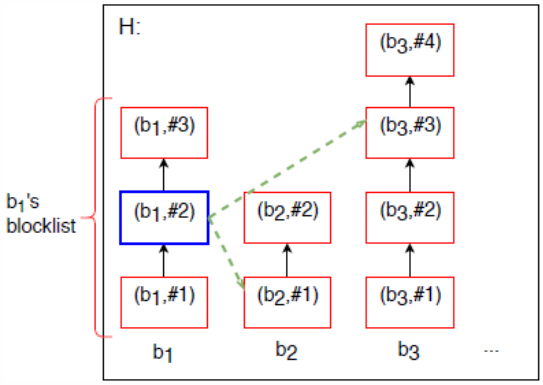

Fig. 2. History. The block from Fig. 1 appears with blue edges. The dashed arrows describe the blocks it references.

We assume a genesis history, H_0, that practically stamps the money and transfers it to predefined accounts (among others, it can transfer money to users of currently popular coins, according to their balance at a predefined point). In H_0 all the banks' blocklists are empty except for a special bank whose account number is all zeros. This bank's blocklist contains a single block (the genesis block), with the "genesis transactions" that transfer money to the predefined user and bank accounts.

There are several rules that a history H should follow in order to be considered `valid'. To define these rules we first define when a given history is a prefix of another. Given two histories, H_1 and H_2, we can naturally define that H_1 is a prefix H_2 if every item in the tuple of

H_1 (a blocklist) is a prefix of the corresponding item in H_2. Clearly we want H_0 to be the prefix of all other histories. Next, recall that each block encodes a history (that is used to compute the current account balances). We want to make sure that a block that appears in H encodes history that is a prefix of H (as otherwise it references blocks that don't exist in H) and that is prefixed by the histories encoded in that block's ancestors (or otherwise that bank's blocklist is not coherent). Finally, the rules for validity of a history H are:
- H_0 is a pre_x of H.
- The first block of each bank in H must reference the genesis block (this might be applied implicitly).
- The history encoded by a block B, that appears in H, must be a prefix of H, and prefixed by the histories
that are encoded in the blocks that come before B in the blocklist of B's bank.
- There should be no circular dependencies between the different blocks of H.
- All the blocks that appear in H contain legal transactions (i.e., they are legally signed, and the relevant accounts have the money to spend, according to the history that the relevant block encodes).

In the rest of the paper we assume only valid histories. Such histories are not commonly found in the air. They are stored in memory. Whose memory? The banks (and anyone else who wants). Each bank starts from a given history (H0 for the senior banks, and some other histories to newer banks). As time goes by, these histories (in the banks' memories) are evolving, as new transactions are introduced.

Roughly speaking, we want the different banks to share the same history, in order to have an agreement about the account balances. However, they cannot really be the same all the time, and it is generally not a necessity. If you want to know your balance at a given point (e.g. to check if you have enough money to buy something), then all you need to do is to ask your own bank (finally it is your bank that sees if you have enough money to perform transactions or not). The true danger is in double spending. To avoid this danger, we want some "coherence" between the histories that are stored in the different banks along time.

We define $(b; t)$ to be the history that is saved at bank b at time t. Let $H_1 = (b_1; t_1)$ and $H_2 = (b_2; t_2)$ be two such histories from (possibly) different banks, b_1 and b_2, at (possibly) different time points, t_1 and t_2, and let b_3 be another bank (possibly same as b_1 or b_2). Let h_1 and h_2 be the blocklists of b_3 in H_1 and H_2 respectively. Coherence requires that either h_1 be a prefix of h_2, or h_2 be a prefix of h_1 (and this should hold for every three banks b_1, b_2 and b_3, and every two time points t_1 and t_2). Otherwise, it means that b_1 and b_2 see different, possibly conflicting, transactions that are committed by b_3. Note that coherence doesn't promise agreement on the current balances. It promises that no one has done any double spending, which is our true fear in distributed coins.

However, we do want that finally there will be agreement. More formally, if a bank b sees at time t the history $H = (b; t)$, then we want the histories seen by other active banks, starting from some time $T > t$, to have H as a prefix (i.e., starting from time T, they all agree at least up to H). We cannot truly assure it for all the banks, as some banks might fail. However, the clients of those failing banks are expected to draw their money from those banks, and the result is that those banks will loose their voting power, so they will have no more (or at least less) importance.

4. Protocol

The protocol that is presented in this section describes just the basic concepts of agreement in our system. Its true implementation is planned a bit different. Our main goal is keeping coherence, i.e., making sure that there are no different histories among banks (where a difference might imply double spending). We achieve it by upgrading our simple history to proven history, which is basically an history that contains a proof for each block that appears in one of its blocklists. The proof on a block is practically the agreement of the other banks to the issuing of its transactions. Note that coherence has no meaning for malicious banks that don't follow the protocol. Rather, the coherence should be between banks that do follow the protocol. Except following the protocol, there is also importance for "responsive" banks - banks that within a small bounded delay receive messages and respond to them accordingly. We define the set of these banks as the "good banks". The good banks can easily share information between them without adding much latency. This information sharing has big importance in preventing malicious acts. Thus, the main goal of the protocol is maintaining coherence between the good banks.

The protocol starts when a given bank wants to issue a set of transactions (either its transactions or its clients). We name this bank as the "issuing bank". The issuing bank creates a new block that contains the required transactions, and that will become the next block on its blocklist. Before this new block can be accepted by the other banks, it must be assured that there is a consensus on its acceptance. The consensus process includes two signing phases. Each phase starts when the issuing bank sign its new block and sends it to the other banks. The other banks should examine this block, sign it, and send their signatures back to the issuing bank. Recall that each bank has a voting power that is determined by a function of its "private" money, and of its customers' money. Once the issuing bank has gathered the signatures of enough banks, where enough means that the relative voting power of these banks is above a predefined threshold (a majority at the least), it can advance to the next phase.

In the first signing phase, the issuing bank simply sends its newly issued block. The other banks, that receive this block, sign this block and send their signatures back to the issuing bank, possibly together with comments. If the issuing bank wants to use the signatures of the commenting banks, it must apply their comments. In the second phase, the issuing bank sends the block that is shaped by the above comments together with the signatures of the other banks from the first phase. Once the issuing bank has gathered enough signatures from the second phase, it can send its block (the same block from the second signing phase) together with the new (and final) signatures to all other banks, as it now posses a proof that its block should be accepted.

There are several reasons for the "double signing" process. First and foremost, it allows a simple compensation scheme, for the banks that took part in the agreement process. The reason is that in the second signing phase there is in fact an agreement on the first-phase-signatures, and that can be used as an agreement on the compensated banks as well (more on compensation later on). As we should soon see, the double signing also gives us an enhanced security. And finally, it allows the issuing bank an option, before the issuing of the final block, to remove from the block transactions that are conflicting with the knowledge of

other banks (this might happen, for example, when a client asks from different banks to draw its account). Now we shall discuss each phase in detail.

4.1 First signing phase

In this subsection we examine another bank (not the issuing bank) that receives the newly issued block. The receiving bank has to verify this block's correctness. We name this bank as a "verifying bank".

Recall that the new block contains a serial number, according to the block position in the blocklist of the issuing bank. The verifying bank first checks that the serial number of the received block is exactly one above the serial number of the last block of the issuing bank blocklist (as much as it aware of). If the verifying bank is missing some blocks in the middle, it asks them from the issuing bank (or any other source). If it already has a different block from the issuing bank with the same serial number, then it now possesses a proof that this bank is malicious, as it tried to publish two parallel blocks. More about the case of malicious banks later on. Let's assume that the verifying bank holds the previous block of the issuing bank. Recall that each block encodes a specific history (using pointers to other blocks). The next thing the verifying bank does is making sure that the history encoded in that new block is prefixed by the history that is encoded in the previous block of the issuing bank. Otherwise, the issuing bank is, once again, considered malicious, as it tried to go back in time. Afterwards the verifying bank compares the history from the new block to its own known history. It verifies first that the two histories are coherent, i.e., that there are no conflicts between them. Then, if the verifying bank misses some blocks, it tries to complete them. Once the verifying bank has all the blocks, that means that the history from that new block is a prefix of its own known history.

After finished with verifying the history, the verifying bank examines the transactions and makes sure that the relevant account balances remain non negative after applying those transactions. After that, the verifying bank could simply sign this block and send its signature back to the issuing bank. However, in order to assure that all bank histories remain updated, the verifying bank attaches to its response an encoding of its own history. If the issuing bank want to use the verifying bank's acceptance for its new block, it must update its history so that it is prefixed by the history encoded in the response of the verifying bank.

At this point the issuing bank starts receiving the signatures of those verifying banks (usual banks that received its block and verified its correctness). Finally, when the issuing bank have the signatures of enough verifying banks, and when it has updated with the required new blocks, it can advance to the next phase. We mentioned that "enough" signatures means that the relative voting power of these banks will be above a predefined threshold. However, the relative voting power of a bank is a function of the history (observing a given history, we can compute the relative power of each bank). A different history shall yield a different voting power distribution. Thus, the history that is used for the computation must be accepted. The relevant history in our case is the history that is encoded in the newly issued block.

4.2 Second signing phase

After the issuing bank has gathered enough signatures from the first phase, it constructs its final block, which is basically the original block, possibly with newer history encoded inside it, and possibly with cancellation of some transactions (as we shall see later on). It signs this final block and sends it to all other banks. When another bank receives such final block, it has to check its correctness. Let's call such bank, that receives that final block, a "confirming bank". The confirming bank makes sure that (1) the transactions in the block are valid, (2) the attached signatures (from the first phase) are valid, and (3) that the set of banks that signed that block (on the first phase) form a majority (according to the history that is encoded in that block). While the confirming bank could be satisfied by the correctness of the block and its proof, it should, however, do one more check before it signs it. The confirming bank should make sure that the other banks received the same block as well (and in particular didn't sign on another block in the first signing phase). This is important in order to make sure that the issuing bank doesn't try to double spend by giving one block to some banks and another block to others. By checking with the other banks in advance, we make sure that none of the "good banks" (banks that follow the protocol and that receive messages from other banks within a small bounded delay) can be used in such a scam, as they all must see the same block or otherwise they will find out that the issuing bank is malicious, before singing for the second time on its block. Thus, to perform such scam, the issuing bank must truly have most of the money placed in banks that are in his own hands, so it can achieve a majority without the help of the good banks.

So, a confirming bank should send a message to the other banks and wait for a reasonable time in order to allow the other banks to respond (a reasonable time should be as minimal as possible, just to ensure standard communication and computation times of a responsive server). If within this time period no other bank reports any problem, it should sign that block. On the contrary, if there is another bank that sends it in return a parallel block - a block of the same bank with the same serial number but with different content, then the confirming bank must not sign that block. Moreover, there is in such case a proof that the issuing bank is malicious (as there are two parallel blocks, both signed by that issuing bank).

4.3 Accepting a new block

Once the issuing bank has gathered enough signatures in the second phase, it can advance to the next phase, which is practically sending the final block (the same block that was sent in the second signing phase) for acceptance. The set of signatures from the first phase (of the "verifying banks") together with enough signatures from the second phase (of the "confirming banks") provide the true final proof for validity of this block. That proof is sent together with the final block.

A bank that receives such block makes sure that the block is valid and contains all the required signatures, and then it can attach it to its known history. We call such bank an "accepting bank". Once the accepting bank has accepted such block, the acceptance is final - it cannot revert it. The reason is that such acceptance signals the users that receive money from transactions that are found in this block, that they indeed received their money. In return, they might have given the corresponding payees other type of goods. Reverting the

acceptance of such block means that those users have lost their money. For this reason, the accepting bank must be sure that this block should truly be accepted.

The rules for accepting a block are as follows:

1. If the accepting bank has confirmed this block in the previous phase, then it must accept it.
2. Otherwise, if the accepting bank has received another parallel block before that, it must not accept that new block.
3. If both of the above cases do not apply, then the accepting bank should check with the other banks if they received another final block that contains all the required signatures. The accepting bank should accept this block only if it receives no evidence of such other parallel fully-signed block.

Following these rules will help once again to guarantee that all good banks accept with each other. More precisely, it assures that every pair of two good banks shall never accept two different parallel blocks. Proof: If a good bank, $bank_1$, accepts the block B_1, then it follows from the protocol that B_1 must be the first block it sees. Thus, there cannot be any other good bank that confirmed a different block, B_2, as it would have send B_2 to $bank_1$, and receive in reply that $bank_1$ has seen B_1. So, the only way such another good bank will accept B_2, is if it receives it with all the required signatures, before it receives B_1. In such a case we can be sure that $bank_1$ didn't confirm B_1, as it should have either hear about B_2 from that other bank, or otherwise that other bank should have received B_1 before B_2. Thus, $bank_1$ accepts B_1 only because it received it with all the required signatures. In this case $bank_1$ has asked the other banks if they received another fully signed block, and they replied that they didn't, so there cannot be such good bank that had accepted B_2.

The downside of the presented protocol is that there might be a chance for a "gap" between the good banks: A malicious bank might introduce two parallel blocks in such a way that part of the good banks will accept one of these two blocks (as they won't see the other block until it is already too late), while the rest won't accept any of these blocks (as they will see the two parallel blocks before they confirmed or accepted one of them). Note however that such a scheme is possible only if the malicious bank can form a majority without the good banks. We prove the last sentence: The good banks will never confirm two parallel blocks as they ask each other before confirming a block. Now, assume that a majority cannot be formed without the aid of the good banks. The result is that a malicious bank cannot produce two fully signed parallel blocks, but at most one. Assume that it has produced such fully signed block, B_1, and another block which isn't fully signed, B_2. The good banks that confirmed B_1 will also accept it (and there must be at least one such good bank that confirmed it). The good banks that didn't confirmed B_1, couldn't have seen B_2 before it, as otherwise they would have warned the confirming banks, and they in return wouldn't have confirmed B_1. Thus, when the non-confirming banks receive B_1 with all of its signatures, they will accept it.

4.4 Compensation

Last but not least is the compensation scheme. Recall from the introduction that a commission will be taken from each accepted transaction, and will be divided between the issuing bank and the other banks that took part in the agreement process for the block that

contains this transaction. Moreover, in order to encourage cooperation between the banks, the commission will be bigger as more banks take part in the agreement. I.e., the commission will be a function of the total voting power that took part in the agreement process. Note, however, that there must be an agreement concerning the identity of the compensated banks (and concerning the size of commission they get). The only agreement that we have is concerning the banks that signed in the first signing phase. For this reason, the compensated banks will be exactly those first-phase ("verifying") banks.

4.5 Additional implications

So far, our protocol gives us a cryptocurrency that is safe and simple (as long as there is enough power in the hands of "good" banks, such that a block cannot be accepted without the confirmation of at least one of these good banks). The world could be simple and safe, but of course there are some complications. The main worries are the following:

1. The possibility for a bank to stop functioning.
2. When a bank goes malicious.
3. Majority coalition ignores the others.
4. When too many banks stop functioning, in such a away that the remaining banks cannot form a majority.

We shall start talking about these issues one by one:

1. If a bank goes down (stop responding), we want its user accounts not to "go down with the ship". Another fear is a bank that ignores some of its user requests. For these purposes, the user can address another bank and ask it to draw his account from his original bank. The new bank will put such request in its next block. The problem is that such requests are dangerous, because a malicious user might send such requests to different banks, and maybe even to keep issuing transactions from his original bank at the same time. Thus, a bank that publishes in its block such user request to "draw" the user's account, might receive "refusals" when trying to commit its new block, if other banks have also received requests from this user. Note that these refusal only concern the problematic transactions. If this issuing bank wants to use the signature of those banks, it simply has to tell in its final block (the block it should send in the second phase) that it has canceled the problematic transactions.
2. There are several cases in which a bank is identified as malicious by the other banks. In its most compact form, maliciousness includes performing an action that cannot be identified as invalid by its own, but only by performing another parallel action. For example, if a given bank issues two blocks with the same serial number but with different transactions (transactions that are valid for themselves), then a bank that receives only one of these blocks cannot see any problem with it. It is the two parallel blocks that create the problem. As such an action is harder to observe (you must have the two blocks), then it put more risk on the system. On the other hand, a bank that issues a block with invalid transactions doesn't put much risk on the system as anyone that receives its block will immediately consider it as invalid and will ignore it. Whether the latter bank should be considered malicious or not is left as a decision to the system planner.

A malicious bank will lose all of its customers and will lose its private money. More precisely, when a bank is recognized as malicious by the other banks, the other banks won't agree to any future requests of the malicious bank, and its private money counts as zero in future votes. As a result, that bank cannot use its private money, and its clients won't be

able to address it in order to perform transactions, and they must address another bank to draw their account into its.

It all starts when a given bank identifies another as malicious. Such bank should put the evidence for the maliciousness in its next block. Every bank that accepts that block (with that evidence), or that put such evidence in its own block, must not sign on any future block of the malicious bank. However, it might be that other banks have already signed a new block of that malicious bank before they received that evidence, and they might have also accepted that new block. Thus, accepting a fully signed block of this malicious bank is still possible, following the rules of accepting a block. At all events, as long as achieving majority is impossible without the aid of at least one "good bank", that malicious bank won't be able to produce new blocks.

3. A majority coalition (or one very strong bank) might decide to ignore all other banks. Such act will prevent other banks from carrying out their client requests, and eventually their clients will probably leave them and join that coalition, which will make the coalition even stronger. We try to avoid that by giving bigger commissions when the majority is stronger, so at least for a while it will give this coalition lower profits, at least up to the point where the users will start dropping from the other banks and join the banks in this coalition. Note that this coalition must have a majority. They cannot use other fair banks, as the signatures they shall receive from those banks will force them to accept blocks of other banks as well (recall that if an issuing bank wants to use a signature, it must get updated with the history that is attached to that signature).

And finally, if such majority coalition truly exists, then the other banks have a final resort – forking, as will be described next.

4. Now we have got to the last case, where advancement cannot be made, as there is no way to achieve a majority. In this case we are forced to use our one last resort – forking. In a fork we have a set of banks (the forking banks) that agree on a given history. This history dictates that banks that are not part of the forking banks are counted as "malicious banks", i.e., they will loose all their private money, and they won't be able to perform any action (as the forking banks will ignore their requests). The result is that the forking banks shall hold all of the voting power.

In the case where there is a set of non functioning banks, forking is rather simple – all active banks agree on the fork. However, in the case of majority coalition that ignore the others, the small banks will probably agree on forking, and the majority coalition will either wait for the other users to ask them to draw their accounts, or otherwise will fork themselves. In such cases, there will be two separate active systems, where each user can spend his money in both. The users don't truly double their money, as the value of the money in both of the systems together will be probably lower than the original one. The banks, however, can probably spend their money only in one of the two systems.

Note: A fork creates a completely new currency. Such fork must not be accepted in some sort of automated software. Moving to a fork must be a fully intentionally and deliberate action.

5. Conclusions

In this paper we presented the new coin Unity. From the user side, its usage is much like every day money, that is stored/deposited in the bank. From the "banks" side, it is a

distributed coin whose consensus is based on the coin possession. By moving away from the traditional single blockchain, this coin can achieve low latency in committing transactions. Unlike other coins, where there is an importance for the passage of time, our protocol is not dependent on any time value, and the only delay that exists is due to messages transmission time and the time it takes to perform the basic required computations of validating/applying signatures. Under these settings, the delay should be on the order of seconds - just like using an everyday credit card. The result of using the protocol is also an increased transactions throughput. However, it is still recommended to combine it with other existing ideas of reducing the load by using "side channels" to perform frequent and small money transfers (micro-payments).

Concerning the presented protocol, note that Byzantine Fault Tolerance protocols usually require that a 2/3 of the players will be decent. This way, for every two blocks there must be at least one decent player that has accepted both, so no parallel block can be accepted. By using the concept of "good banks" we require much less – that there won't be a majority without the good banks.

Last word concerning the small latency, note that the communication between banks is a direct communication - each bank should know the IP of the other banks. This is not a must (there is no importance from where a message signed by another bank has been received from, as long as it is properly signed), but it can save important time, and there is no reason why it won't be that way.

5.1 Future Work

The protocol that appears in this paper describes just the basic principles of the presented approach. Its implementation is planned a bit different. In the implementation, every action such as signing on a new given block will appear also inside a block (your own next block). More precisely, by simply pointing in your next issued block to another newly constructed block, it is interpreted as if you have just signed that block. In such a scheme the history encoded in a bank's "signature" on another block is implicit – it is just the history encoded by the block where that "signature" (a pointer to the verified/confirmed block) is found. As the "signatures" appear inside the blocks, the result is that the "proofs" for validity of blocks in a given history are part of the history itself. Note that this scheme requires that you won't have to wait for your block to be accepted before issuing the next block. The result is that there will be part of the history that is "accepted" and part that is "under acceptance process".

In this paper we assumed a completely transparent system, where every transaction that is accepted is visible to the public, so that everyone can see the source and destination accounts, and the sum of money that was transferred (just like in Bitcoin). Yet, the identities of the account owners are of course generally unknown. An interesting direction is if we can limit the transparency to the level of banks, so that the only one that truly know both the transaction source and destination is the bank of the issuing/receiving client.

Another direction for future work is on the concept of using parallel blockchains. Such parallel blockchains are in fact parallel dependent SMRs. I.e., we have parallel replicated state machines where each state ma- chine makes its own advancement, but where there are operations that might conflict. We use it here in a less permission model, but it can be also introduced to a simple model where the identities of the players are well known.