Democoin:
A Publicly Verifiable and Jointly Serviced Cryptocurrency

Sergey Gorbunov* MIT

Silvio Micali† MIT

May 30, 2015

A Quick White Paper

(This technology is covered by three patent applications)

Abstract

We present a new, decentralized, efficient, and secure digital cryptocurrency, in which the ordinary users themselves keep turns to ensure that the systems works well.

*sergeyg@mit.edu

†silvio@csail.mit.edu

1 Introduction

Money has been around for thousands of years. In the past, it was very physical, as in the case of gold bars or coins. However, with the emerge of computer and network technologies, electronic forms of money and payment systems have been considered. (See, in particular, Chaum's electronic cash [2]. In principle, money can be made totally electronic. If each money transaction goes through a single, trusted, central authority A, this authority could keep track and publish, at each time t, how much everyone owns and who owns to whom. On one side, this approach to money has the big advantage of being very efficient for users, in that the public record kept by A can be very compact and easy to consult, and yet sufficient to enable users to make payments to one another with confidence. On the other side, however, this centralized approach also has severe limitations. In particular, it is hard for a large community of users to find an entity A that everyone trusts. This, in the eyes of many, continues to be true even if A were chosen to be a government. For example, the authority A could wipe out a user U's ability to pay by simply publishing that he/she/it no longer owns any money, or could make it appear that U has made payments to someone else that U never made. Thus, this centralized approach could dramatically fail if A becomes corrupt, or is compromised by adversaries.

To avoid this weakness, cryptocurrencies like Bitcoin have been created that are very decentralized [5]. However, these systems require a public file ("ledger") that is very big and very inefficient to maintain and update. As a result, these systems too may not be too useful, particularly if the number of users and transactions grows. In addition, Bitcoin requires that a very large majority of key players are honest.

It is thus the purpose of the present invention to put forward an electronic money system that has the advantages of the centralized approach, but without trusting the central authority that maintains the public record of transactions, and without suffering the inefficiencies of the decentralized approach.

We call our approach *Democoin* in order to convey that the power to ensure that the system works properly resides with the players themselves. In order to better appreciate the advantages of our system, it is useful to quickly recall our background.

2 Background

2.1 Cryptographic Primitives

Digital Signatures. A *digital signature scheme* consists of three fast algorithms: a probabilistic *key generator* G, a deterministic *signing algorithm* S, and a *verification algorithm* V.

Given a number k as an input (e.g., $k = 4,000$), a player x uses G to produce a pair of k-bit keys (i.e., strings): a "public" key PK_x and a "secret" signing key SK_x. Crucially, a public key does not "betray" its corresponding secret key. That is, even given knowledge of PK_x, no one other than x is able to compute SK_x in less than astronomical time.

Player x uses SK_x to digitally sign messages. For each possible message (binary string) m, x runs algorithm S on inputs m and SK_x in order to produce a string, denoted by $SIG_x(m)$ or SIG_{PK_x}, referred to as x's digital signature of m, or a digital signature of m relative to PK_x. Without loss of generality, we assume that m is retrievable from $SIG_x(m)$. (After all, a digital signature of m could always include m itself.)

Everyone knowing PK_x can use it to verify the digital signatures produced by x. Specifically, on inputs (a) the public key PK_x of a player x, (b) a message m, and (c) an alleged digital signature of x for the message m, the verification algorithm V outputs either YES or NO, so as to satisfy the following properties:

1. *Legitimate signatures are always verified:* If $s = SIG_x(m)$, then $V(PK_x, m, s) = YES$; and
2. *Digital signatures are very hard to forge:* in essence, without knowledge of SK_x finding a string s such that $V(PK_x, m, s) = YES$, for a message m never signed by x, requires astronomical amount of time.

Accordingly, to prevent anyone else from signing messages on his behalf, a player x must keep his signing key SK_x secret (hence the term "secret key"), and to enable anyone to verify the messages he does sign, x has an interest in publicizing key PK_x (hence the term "public key").

Collision-Resilient Hashing. A collision-resilient function H quickly maps arbitrarily long strings to strings of fixed length (e.g., 256-bit strings) so as to guarantee that finding two different strings X and Y, such that $H(X) = H(Y)$, requires an astronomical amount of time.

Collision-resilient hashing functions are often used within digital signature schemes. If, for instance, the schemes can only sign messages consisting of at most 4,000 bits, and a player x wishes to sign a longer message m, then he may sign $H(m)$ instead. That is, $SIG_x(m)$ could be defined to consist of $SIG_x(H(m))$, or of the pair $(m, SIG_x(H(m))$, in order to ensure that m is retrievable form $SIG_x(m)$.

2.2 Bitcoin

At a high level, in Bitcoin and its variants [4, 5, 6], at every point in time, (the owner of) a given public key PK owns a given amount of money. Some of this money can be transferred from PK to another public key PK' via a digital signature, computed with the secret key corresponding to PK. Lacking trusted central authorities, this signed transaction, T, is broadcasted over the web in a decentralized manner. That is, any one who sees T will forward it to his neighbors, who will forward to their neighbors, and so on. Everyone seeing T is responsible for verifying its validity. This verification certainly includes verifying that the digital signature proper is valid. However, it must also include verifying that the key PK had sufficient money to transfer.

The latter verification may require validating the entire transaction history. Note that this is no trivial task if the number of total transactions is large! Moreover, since one cannot guarantee that everybody has seen all transactions, it becomes necessary to achieve consensus on what the current transaction history is. To simplifying this tasks, transactions are aggregated into blocks, $B_1, B_2, \ldots$. Each block includes the hash of the previous block, a collection of new transactions, and a solution to a cryptographic riddle. Such a riddle depends on the previous block and the new transactions.

A user seeing blocks B_1 through B_k, and a collection of new (and valid) transactions, tries to aggregate these transactions into a new block B_{k+1} by solving the proper riddle. He is incentivized to do so because, if he succeeds in generating the B_{k+1} block before anyone else, then he will will gain a monetary reward. Cryptographic riddles are sufficiently complex to solve. A single user may take a very long time to solve a given riddle. However, there are many users trying to produce a new block and thus to solve a riddle. Currently, the riddle complexity is chosen so that, in expectation,

it takes 10 minutes for some user to find a solution. Ten minutes is ample time for every one to see a new block and thus gain consensus on what the transaction history is. Nonetheless, the possibility exists that two users solve the riddle sufficiently simultaneously. For instance, having seen the same block chain $B_1, \ldots, B_k$, one user may succeed in producing a new block B'_{k+1} and, almost simultaneously, another user may succeed in producing a block B''_{k+1}. In this example, each of the two users may broadcast his own new candidate block in an attempt to get the associated reward. At some point later, when even more transactions are being generated, a third user U may see two possible chains $B_1, \ldots, B_k, B'_{k+1}$ and $B_1, \ldots, B_k, B''_{k+1}$. To create a new block B_{k+2} and collect the associated reward, U needs to decide whether to try to solve the riddle associated with the new transactions and either block B'_{k+1} or block B''_{k+1}. Even if he does this in parallel, because solving a riddle requires a computational effort and because the reward goes to the first user who generated the block B_{k+2}, U will broadcast the first solution it finds in order to secure this reward. Accordingly, when he does so, some of the users will see a $k+2$-long block chain $B_1, \ldots, B'_{k+1}, B_{k+2}$, and others will see a chain $B_1, \ldots, B''_{k+1}, B_{k+2}$.

Because users are asked to append to the longest chain, the $k+1$st block will eventually become unique. In practice, although the last (or even the second last) block one sees may change, one can safely assume that the first k blocks in a chain of length $k+2$ will no longer change. Accordingly, if a transaction, belonging to the third last block, transfers an amount X from public key PK to public key PK', then the owner of PK' 'can consider himself paid'.

Bitcoin Statistics. We present some basic statistics about Bitcoin protocol referring to blockchain.info [1], to understand some of the scalability issues associated with it. Since Bitcoin is fully a peer-to-peer protocol, *every player* must store the entire public ledger of transactions. As of February 2015, the size of the public ledger exceeds 28 GB, which grew from 5 GB two years ago. The number of current public keys is only about 225000, which grew from 50000 two years ago. Moreover, the total number of transaction at every unit of time (i.e. 10 minutes) is only 650, while 2 years ago it was only around 450 transactions. Extrapolating, even if the current public ledger can fit on *some powerful* cellphones, in less than two years from now, this might no longer be feasible.

Also, since in Bitcoin protocol every player must use its precious computational cycles on solving certain cryptographic puzzles, the total computational power of all Bitcoin players combined recently broke 1 exaFLOPS [3]. (exaFLOPS refer to the number of Floating-point Operations a computer can do Per Second. Or more simply, how fast a computer can tear through math problems. An exaFLOPS is 1018, or 1,000,000,000,000,000,000 math problems per second). Top 500 most powerful supercomputers can only muster a mere 12.8 percent of the total computational power of the Bitcoin players.

Weaknesses of Bitcoin The above discussion suffices to highlight some of the inefficiencies of Bitcoin (and its variants). Namely,

- *Large Storage.* Users must download and store a large transaction history.
- *Computational Waste.* To add a new block to the public ledger requires a large amount of computational resources in order to solve the necessary riddle, not only for the user that succeeds, but also for all others that tried but failed.

- *Payment Time.* 30 minutes (or more) are needed to be sure that one is paid in Bitcoin. Assume that the owner of a public key PK pays an amount X to the owner of a public key PK' by generating the necessary digital signature at time t. Then, to be sure that the transaction is agreed upon by everyone in the system, the owner of public key PK' must wait 30 minutes. Indeed, since, on average, it takes about 10 minutes for this transaction to appear in a new block and another 20 minutes for this block to become the third last one. One may engineer things so that the expected time to add a block is lower than 10 minutes, but then this saving might also require waiting for a block to be more than 3-deep in order to be reasonably confident that the transaction history will no longer affect that block.

3 Three Alternative Systems

To avoid the weaknesses of Bitcoin, we explore three alternative currency systems. We start with a centralized alternative, *CentralCoin*, and highlight some technical ideas that will prove useful also in subsequent alternatives. We then move on to *SpreadCoin*, a more decentralized system. The responsibility for the integrity of this system is spread across a multiplicity of specialized entities. Finally, we introduce *DemoCoin*, where the responsibility for keeping the integrity of the system resides with regular users.

Players. A player is a collection of individuals, an entity, or a collection of entities. A player i may have one or more public keys, and may be identified with one of these public keys. For instance if PK_i is the public key of a player i, we may also talk of player PK_i.

Money. In our systems, money can be denominated in US dollars, in another existing currency, or in a currency of its own. We denote a monetary unit by the symbol #.

Time. We envisage a time sequence, $T = 0, 1, 2, \ldots$ For simplicity, the time interval $[t_i, t_{i+1}]$ is the same for all i (e.g., two minutes or one minute), but can be adjusted dynamically depending on the number of players, transactions, etc. Preferably, the time unit is chosen so that, despite reasonable clock shifts, most (or all) players know what the current time t is.

Payments. Money is associated to individual public keys. Initially, some public keys are publicly known to have some given amount of money. Money is transferred, via a digital signature, from one public key to another. (This is for clarity only. To transfer money to, say, 4 public keys, one simply executes 4 separate transfers. But one can also envisage multiple money transfers by means of a single transaction.) For instance, a payment P of an amount A from a key PK to a key PK' at a time ($time$) might be

$$P = SIG_{PK}(PK, PK', \#A, time, I)$$

where I represents any additional information deemed useful, such as an indication of the reason of the payment, or no information. The main problem is to determine whether PK had #A to give.

4 Centralcoin

In Centralcoin, a special player, the *central verifier*, CV, is responsible to verify which money transfers are valid and to report compactly the state of the system, and *cannot cheat without publicly proving his guilt.* The public key of CV, PK_{CV}, and his url are publicly known.

Centralcoin works in rounds. Each round t conceptually consists of three phases (e.g., 20 seconds each), and is completed within the time interval $[t, t+1]$ (e.g., in one minute). We recall that at the beginning of the protocol (that is, at time $t = 0$) all players know a compact list of public keys with pre-loaded amounts of money.

Phase 1.

All players download the two CV-signed lists of the previous round, PAY_{t-1} and $STATUS_{t-1}$; verify CV's digital signatures; and verify that $STATUS_{t-1}$ was updated correctly from $STATUS_{t-2}$ and PAY_{t-1}. (Alternatively, a player may only verify a subset of the status report, corresponding to his own public keys.)

Phase 2.

Each player generates his own round-t payments and causes them to become available to CV.

Phase 3.

CV computes, digitally signs, and publishes (e.g., at a given url) the new lists of the current round: PAY_t, which specifies all valid payments of round-t, and $STATUS_t$, which specifies the account information at the end of round t. For instance,

$PAY_t = SIG_{CV}(t; P_1, P_2, \ldots)$[1] and $STATUS_t = SIG_{CV}(t;\ (PK_1, \#A_1, I_1); (PK_2, \#A_2, I_2); \ldots)$[2]

Discussion

- *Ultra Compact Record.* The big advantage of a centralized system is that the full status of the system is very compact.
- *New Public Keys.* A new public key PK' may enter the system via a payment $(PK, PK', \#A, I)$ appearing in PAY_t at some round t.

  (Alternatively, CV —or a separate entity— may register new keys, who first appear with a balance of 0 in $STATUS_t$ at some round t.)

[1] Recalling that a payment P is of the form $P = SIG_{PK}(PK, PK', \#A, time, I)$, the list is preferably ordered by the paying public key first, by the paid public key PK' second, and by the amount A third.

A payment is valid if the signature of the paying key is valid, and if the amount is valid, relative to the amount of money that PK has at the end of round $t-1$ and all the preceding round-t payments of PK. For instance, if, according to status S_{t-1}, PK has $\#A$ at the end of round $t-1$, the first k payments (sorted by *time*) of PK in this round have a valid signature of PK and have a total amount of $A' < A$, while the amount of the $k+1$st payment of PK is greater than the remaining balance $A - A'$, then the $k+1$st payment of PK is not considered valid.

[2] where

PK_i is the ith public key in the system, in lexicographic order,

$\#A_i$ the amount of money PK_i has, I_i is any additional information about PK_i, and

n_{t-1} is the total number of public keys at time $t-1$.

- *Tree-Hash-and-Sign.* Authenticating the lists PAY_t and $STATUS_t$ is very efficient, as CV needs to compute a single signature for each list. However, a player wishing to keep an authenticated record only for a single payment at round t, would have to download the entire PAY_t. Similarly, a player wishing to keep an authenticated record for the balance of money available to a given public key at the end of round t would have to download the entire $STATUS_t$.

  To lessen this burden of such a player, CV may digitally sign each entry in PAY_t or $STATUS_t$. In this case, however, it may be challenging for CV to produce so many digital signature within a single round phase. The best is to to have the CV to tree-hash (rather than simply one-way hash) the two lists, and then digitally sign just the root of each hash tree. The big advantages of this approach are that:

  1. CV can authenticate each entire list by means of a *single digital signature* and *one ordinary hash for each entry in the list*;
  2. The digital signature authenticating the list is *compact* (i.e. independent on the number of entries in the list).
  3. Someone interested only in the authenticated record about a given item in either list may have to deal only with a minimal amount of data.

  Details about tree-hash-and-sign are presented in our final section.

- *Semi-Honesty.* In Centralcoin, CV need not be totally trusted, but *semi-honest*, that is, *refraining from any dishonest behavior resulting in a publicly available proof of his guilt.* In particular, lacking knowledge of the secret key SK corresponding to a given public key PK, CV cannot manufacture a payment from PK to some other public key PK'. Nor can CV, without being caught, illegally diminish the amount of money PK has. Assume that CV does so, for the first time, at some round t. Then, CV has already correctly signed the correct $STATUS_{t-1}$. The only legitimate way of decreasing the amount of money of PK in $STATUS_t$ consists of subtracting all the amounts that PK has paid to other keys in round t (and adding all the payments PK received in round t). Thus, since these transfers are digitally signed by PK, by illegally lowering the money available to PK, CV must digitally sign something false, generating a public *proof* of his guilt and exposing itself to possible high fines or other sanctions.

- *Simultaneous Electronic Transactions.* CV may still, however, prevent money to be transferred from some key PK to some other key PK'. That is, CV may prevent the owner of PK (respectively, PK') to use (respectively, receive) any money. In fact, although timely receiving a valid round-t payment $P = SIG_{PK}(PK, PK', \#A, t, I)$, the CV may not include P in PAY_t. Yet, it may be hard for the owner of PK (or PK') to prove that he indeed did give P on time to CV. It is a question of whom one believes. To keep CV unambiguously accountable for this type of cheating, we propose to utilize the technology of US Patent 5,666,420, Simultaneous Electronic Transactions, and foreign equivalents. In essence, this technology guarantees the exchange a message for a receipt so that (a) the recipient of the message learns the message simultaneously with (b) the sender getting a corresponding, very detailed, and digitally signed receipt. In this application, the message consists of the payment P, the receiver is CV, and CV cannot learn P without also signing a receipt for P. In this fashion, whether the sender were the owner of PK or the owner of PK', or someone acting on their behalf, CV could not, with impunity, ignore the payment P. Indeed, CV must produce a digital signature proving that it

received P on time, and thus proves its own guilt if it does not include P in PAY_t, a list that it itself digitally signs.

With simultaneous electronic transactions, Centralcoin is free of undetected cheating, and with tree-hash-and-sign Centralcoin guarantees efficient storage and retrieval of even individual authenticated records. However, the system may be prone to sabotage. Indeed, CV is a unique point of vulnerability.

5 Spreadcoin

In Spreadcoin, there are multiple *verifiers*, $V_1, \ldots, V_k$. Each verifier V_i has his own public key, VPK_i, and its corresponding secret key, VSK_i. Each VPK_i, is assumed to be publicly known. Preferably k is odd: e.g., $k = 11$. Spreadcoin only requires that *a majority of the verifiers are semi-honest*. Spreadcoin works in rounds as follows.

Phase 1.

All players download the authenticated lists PAY^i_{t-1} and $STATUS^i_{t-1}$ produced by each verifier V_i in the previous round, and checks the associated signatures of the verifiers.

Phase 2.

Each player provides his own round-t payment to the verifiers.

Phase 3.

Each verifier i computes and publishes (e.g., at a given url) two authentiticated lists: PAY^i_t, specifying all valid payments of round-t, according to i, and $STATUS^i_t$, specifying the money balance of each public key at the end of round t, according to i. For instance,

$$PAY^i_t = SIG_i(P^i_1, P^i_2, \ldots) \quad \text{and} \quad STATUS_t = SIG_i(t;\ (PK^i_1, \#A^i_1, I^i_1); (PK^i_2, \#A^i_2, I^i_2); \ldots)\ .$$

- The round-t list of valid payments, PAY_t, is taken to be the list of all payments P that, *for a majority of the verifiers i,* belong to PAY^i_t.
- The round-t account status, $STATUS^i_t$, is taken to be the list of all tuples $(PK, \#A, I)$ that, *for a majority of the verifiers i,* belong to $STATUS^i_t$.

Discussion

- *Efficiency.* The record continues to be very compact. Handling 650 transactions every 10 minutes, and 275000 keys (as in Bitcoin) is no problem. Even with a million public keys and a one million transactions every minute, it would be much preferable to Bitcoin's public ledger.
- *Semi-honesty.* Simultaneous electronic transactions could again guarantee that each verifier is kept fully accountable for all his actions. As for CV in Centralcoin, a verifier in Spreadcoin cannot act dishonestly without also generating a publicly available proof of his own guilt.
- *Resilience.* Spreadcoin is much more resilient than Centralcoin. To subvert Spreadcoin, an enemy would have to successfully bring down 6 of 11 sites, if $k = 11$, or 51 out of 101, if $k = 101$. Moreover, as the majority is what counts, one is less pressed to pursue occasional verifiers who misbehave, despite the fact that they are going to be caught.

- *Pre- and Post-Facilitators.* To avoid a player to send or cause to send his own payments to multiple verifiers, we may envision that he sends them to a facilitator, who then distributes them to all verifier. (Or that he posts them where the verifiers can pick them up.)
  Similarly, a facilitator can collect PAY_t^i and $STATUS_t^i$ from every verifier i, and make all of them conveniently available in a single location, or even compile (and digitally sign, so as to keep himself accountable) the proper lists PAY_t and $STATUS_t$.
  One can also arrange each round into more phases. The first ones are for allowing a player to use the facilitator. The other ones are for allowing a player to deal directly with the verifiers, if the facilitator does not work satifactorily.

6 Democoin

In Democoin, multiple verifiers are randomly and independently chosen at each round, from an appropriate set of potential verifiers (e.g., the set of all players), in a way that is clear to everyone. The only requirement is that a reasonable majority of the potential verifiers are semi-honest.

Phase 1.

All players download the authenticated lists PAY_{t-1}^i and $STATUS_{t-1}^i$ as in Spreadcoin.

Phase 2.

Each player computes the current-round set of verifiers, and causes his own round-t payments to become available to them.

Phase 3.

Each verifier i computes and publishes (e.g., at a given url) PAY_t^i and $STATUS_t^i$ as in Spreadcoin. For instance,

$$PAY_t^i = SIG_i(P_1^i, P_2^i, \ldots) \quad \text{and} \quad STATUS_t^i = SIG_i(t;\ (PK_1^i, \#A_1^i, I_1^i); (PK_2^i, \#A_2^i, I_2^i); \ldots)\ .$$

PAY_t and $STATUS_t^i$ are defined as in Spreadcoin, but relative to the verifiers of round t.

Preferred Verifier Selection. In the preferred embodiment, each public key PK, appearing in $STATUS_{t-1}$ and belonging to a potential verifier, becomes a round-t verifier key with some small probability p, so that the expected number of verifiers is a suitably chosen integer k.

For instance, let H be a (preferably one-way) hash function, and preferably let v_t be a substantially unpredictable value that becomes public at time t. Then, PK is a verifier key at round t if

$$.H(PK, v_t) \leq p\ .$$

That is, we interpret $H(PK, v_t)$ as the binary expansion of a number between 0 and 1, and PK becomes a verifier key if this number is less than (or equal to) p.

Discussion

- *Semi-honest Majority of Actual Verifiers.* If H is a suitable hash function, then, for every x, $H(x)$ de facto is an uncontrollable random number. Moreover, because H typically maps arbitrarily long values to 256-bit values, the probability that $.H(PK, v_t) \leq p$ essentially coincides with p. The probability p can be fixed, or may depend on the number of users, so that one can ensure that the expected number of verifiers of a round is k as desired.

  Thus, because a reasonable majority of the potential verifiers are semi-honest, *with high probability the majority of the actual verifiers are semi-honest.*

  In addition, besides the fact that any possible dishonest behavior generates a publicly observable proof of guilt any way, dishonest actual verifiers would lack the time to coordinate themselves. This is so because no one could even predict —let alone control— who the verifiers of a future round will be. Indeed, the actual verifiers of a given round t depend on the value v_t that becomes available only at (or close to) round t. For instance v_t could be the (rough) temperature of several main cities around the world at time t, or the latest number of stocks last traded at the NY Stock exchange, etc.

- *Verifier Availability.* Of course, even if well selected, verifiers need to be available in order to perform their vital functions. It is thus important to realize that verifiers have great incentives to be available. Credit cards typically ask for a variable processing fees between 2% or 3% of the payment amount processed, in addition to a fixed processing fee. The verifiers in Spreadcoin or Democoin can also be paid similar fees. For simplicity only, but without any limitation intended, assume that 1% of the amount of each payment P in PAY_t is divided among all verifiers of round t who have processed P, and thus included P in PAY_t^i. Then, the owner of a potential verifier key PK has very strong incentives to be aware of when PK is selected as a verifier key, and thus to process all the round-t payments he can, in order to maximize the money owned to him. With this policy, even if one or two selected verifiers do not perform their duties, there is enough of them who actively process the round-t payments.

  Note that a selected verifier i for round t can contribute his own PAY_t^i and $STATUS_t^i$ in a common pool, from which his signatures are retrieved. Alternatively, recall that each record in $STATUS_{t-1}$ consists of a tuple of the form $(PK, \#A, I)$, and thus the owner of PK can choose to include in the information field I the url at which PAY_t^i and $STATUS_t^i$ can be found if PK becomes the key of verifier i.

  Also note that the information field I can also be used to indicate if PK declines or welcomes the opportunity to be selected as a verifier key.

- *Stability and Consensus.* One could select the verifiers of a given round t in a way that also depends on crucial aspects on previous rounds. For instance, in a way that depends on $H(STATUS_{t-1})$.

  Note, however, that this verifier selection is less stable than our preferred verifier selection. For instance, consider a user U who has been out-of-touch for a few rounds. Such a player may recall —say— $STATUS_{t-5}$, but not of $STATUS_{t-1}$. Yet, because new public keys are added slowly, the set of public keys in $STATUS_{t-1}$ and $STATUS_{t-5}$ are essentially the same. Thus, because our preferred process turns each public key into a verifier key individually and with small probability, the set of true verifiers of round t and the set of verifiers of round t according

to U are essentially the same. Better said, the opinion of the true verifiers, and that of the majority of the verifiers in U's mind will most probably coincide.

In other words, despite the lack of synchrony and centralization, and despite the verifiers may change completely at each round, our preferred verifier selection guarantees a remarkably accurate consensus in a very efficient manner.

- *Reputable Majority.* No matter how good the selection process may be, one may try to subvert it by packing the set of potential verifier keys with keys of "corrupted people". This, however, may be hard to do if the number of players is large. (Much easier would be, in Bitcoin, to control the set of miners, which already belong to a few consortia (i.e., "mining pools") anyway.) Nonetheless, some additional precautions can certainly be taken.

  One possibility is to elicit an entry fee, or a (pro-rated) yearly fee from each public key in the system. Such a fee could be paid outside the system or within the system (e.g., via an automatic payment from each public key to a given key —or some given keys— in the system). This way, packing the set of public keys with artificially numerous and potentially controllable keys would be very costly.

  A second possibility is having the probability of turning a public key PK into a verifier key depend also on the amount of money that PK as at a given round t, that is, on the second field of the $(PK, \#A, I)$ in $STATUS_{t-1}$. This strategy ensures that players who are more heavily invested in the system also have more responsibility in running it. Accordingly, one does not get much advantage by artificially growing the number of his own public keys in the system. What matters is the total sum of money one has in all his keys. Another advantage of this strategy is that the envisaged distribution of processing fees is proportional to the amount one owns. Thus, "no one looses money on average for payment processing".

  A third possibility is having a separate entity, call it the *verifier registration authority* (VRA), who certifies (anonymously or not) the public keys eligible to be selected as verifier keys. In this case, the VRA may ensure that each user has at most one key PK that can become a verifier key. This way, it becomes harder to pack the candidate verifier keys with easily controllable keys. For instance, the VRA may ask a registrant to provide a proof of identity (and possibly insert some indication of this identity) in the certificate for the public key PK: e.g., $SIG_{VRA}(PK, VerifierEligible, I)$. Alternatively, the VRA could authenticate a single list of public keys eligible to become verifier keys.

  A fourth possibility is to have a mixture of verifiers: for example: (a) fixed set of verifiers (possibly none) as in Spreadcoin; (b) a set of dynamically selected verifiers (possibly none); and (3) a set of registered over-time verifiers (possibly none).

- *Efficient Updates for Potential Verifiers.* For logical clarity, we stated that, in Phase 1, all players download the authenticated lists PAY^j_{t-1} and $STATUS^j_{t-1}$ at each round. Let us now argue that it is sufficient for a potential verifier i to download the relevant list(s) only once in a while.

  For instance, assume that i downloads the status-list once a month, and that suddenly, at round t, she is selected to become a verifier. To perform her role, and receive the corresponding reward, i needs to know $STATUS_{t-1}$. Even without any facilitator, once selected, i could immediately retrieve the published and authenticated lists $STATUS^j_{t-1}$, for sufficiently many

verifiers j of round $t-1$. To do this, she needs to determine who the verifiers of rounds $t-1$ were. Since each potential verifier is chosen at round $t-1$ if $.H(PK, v_{t-1}) \leq p$, to make this determination i needs to know two pieces of information: (a) the public keys of the potential verifiers and (b) the value v_{t-1}. Note there is no problem for her to obtain information about the value v_{t-1}, because by definition this value was a publicly available one. As for piece of information (a), i can essentially compute it from her status-list of a month ago. For instance, assume that the number of potential verifier keys grows at the rate of 20%/year, and that there were $500,000$ such keys a month before, when i downloaded the full status-list of the system. Then, roughly 10,000 new potential verifier keys have been added in the last month without i's knowledge. Assume that the selection probability p were such that 101 verifiers were individually and randomly selected at round $t-1$. Then, the probability that one of the newly added 10,000 keys is actually selected as a verifier key is 0.0002.[3] Accordingly, the set of actual verifiers of round t computed by i according to her one-month-old record, coincides, with high probability, with the true list of verifiers of round t, computed based on all potential verifier keys of round $t-1$. Moreover, *with overwhelming probability*, the number of actual verifiers selected from the newly added 10,000 keys will be *very few*. Accordingly, when a reasonable majority of the potential verifiers are semi-honest, with overwhelming probability, the majority of both sets of verifiers (i.e., the actual verifiers, and the verifiers according to i's one-month-old data) will be semi-honest. In sum, even if she downloads the full list of potential verifier keys once a month, a newly selected verifier, would be ready to perform her functions accurately.

Alternatively, rather than performing a full status download once a month, a potential verifier i could download once a month just all transactions of the last 30 days, from which (given her previously computed full status information of a month ago) i can easily reconstruct the current status information. Furthermore, if tree-hash-and-sign mechanism is used, i could easily check that her reconstructed status is correct by simply `(i)` locally computing the alleged root hash of the tree for round $t-1$, and `(ii)` verifying that each verifier authenticated the same root hash at round $t-1$.

Alternatively yet, in order to facilitate these updates, each selected verifier j of a given round t, in addition to PAY_t^j and $STATUS_t^j$, could also authenticate and publish NEW_t^j, the list of newly added potential verifier keys at round t. Notice that, by so doing, j remains fully accountable for all the data she publishes.

- *Efficient Updates for All Other Players.* A player i without verifying responsibilities may only care about whether a given payment P made to him is valid. Such a player can certainly perform monthly updates as a potential verifier. However, with no rewards and no liabilities, he may be satisfied with a much lighter check of the validity of P. For instance, if a facilitator is present, and tree-hash-and-sign is used, then i may download only the facilitator-authenticated information that enables i to determine whether P is a valid payment.

  Alternatively, if i downloads full status information once a month, then he may use his one-month-old information about the set of potential verifiers, and the new public value v_t of round t, in order to compute only one alleged verifier j for round t; and download and verify only the compact j-authenticated record of payment P (assuming again that tree-hash-and-sign is used).

[3] The probability that one the the new 10,000 keys is selected when the target number of verifiers is 11, would be even lower: namely, 0.00002.

- *Democoin vs. Bitcoin.* Bitcoin relies on a very complex and space- and time-inefficient mechanism to ensure that the status of the system cannot be subverted. Moreover, more and more it is being pointed out that a very large honest majority is required for the system to work properly.

  By contrast, Democoin relies on a very simple and very efficient mechanism to ensure that a player with obsolete knowledge about the status of the system can accurately and securely reconstruct the true current status of the system. Moreover, Democoin only requires that a reasonable majority of the users is honest.

7 Scalability and Sample Instantiations

Democoin is very scalable, particularly with the help of a properly designed architecture for information flow. To concretely argue that this is the case, we consider three sample instantiations —referred to as "Urban", "Regional", and "National"— differing in the envisaged number of users and transactions. In all these instantiations, we envisage that

(a) A round consists of 10 minutes.[4]

(b) A single authenticated payment (or status report record) is about 100 Bytes.
(Indeed, 100 Bytes are sufficient to include large amount of auxiliary information.)

(c) A player can efficiently retrieve information about other players necessary for communication.
(E.g., via registry or an IRC channel with players' public keys and IP-addresses information).

(d) Anyone can efficiently retrieve information from a storage provider, *the cloud.*
(E.g., Amazon Cloud acts as a post-facilitator.)

We emphasize once more that in all our sample instantiations, the cloud (i.e., the post-facilitator) is NOT a trusted central authority. Indeed, the cloud cannot fake a payment on behalf of a user, cannot maliciously alter how much money a given public key holds, nor selectively remove from the full status report the information about some players, so as to de facto *deprive them of the use of their money.*[5] *At most, the cloud may not post the entire status report. Should this indeed happen in some round, then no money changes hands in that round, and a new cloud provider can be used. Moreover, even this problem can be mitigated by relying on multiple clouds: for instance, Amazon Cloud and Google Cloud.*

We highlight four criteria for analyzing the scalability of Democoin:

Network Bandwidth: the bytes per second/month a player should be able to transfer.

(Indeed several cell-phone or Internet providers cap the total number of data that a user can exchange in a moth.)

Connection Capacity: the maximum number of simultaneous connections a player can have.

Storage and Computation: the resources needed for a user to participate in the system.

[4]Note that this is the time it takes Bitcoin to generate a new block. (But recall that one would have to wait for three blocks to be generated before have some confidence that one transaction in the third last block will enter the definite transaction history. By contrast, in Democoin, a definite status report is reached after each round.

[5]In fact, a verifier generates a status report by digitlly signing together the money currently held by all public keys. Thus, the cloud may at most refuse to post an entire status report, but not pick and choose which public keys will appear in the report.

7.1 The Urban Instantiation.

The Urban Instantiation is so defined:

- *Number of Users:* 300,000.
- *Number of transaction every 10 minutes:* 1,000.

That is, in the Urban Instantiation, users and transactions numbers are slightly larger than those in the current usage of Bitcoin.

Data Size Since a single payment consists of 100 Bytes, the approximate size of the PAY report is only 100KB, the size of a full status report is about 30MB, and (using tree-hash-and-sign mechanism) the size of self-sufficient authenticating record for a single public key is about 2KB. Altogether, these are very reasonable sizes (and a 30MB status report is certainly preferable to a 15GB public ledger in Bitcoin).

To reduce network bandwidth, connection capacity, and storage we architect a simple, generalizable, and effective information flow, depicted in Figure 1 and described below.

Architecture At each round t, we envisage 110 verifiers (chosen as before), organized in a two-level, 11-node tree: a root with 10 children (and thus 10 leaves in this case). We regard the root to have level 1, and each leaf to have level 2. The verifiers are partitioned in 11 groups/buckets of cardinality 10 each. Each group is conceptually assigned to a separate node in the tree. We refer to the 10 verifiers assigned to the root as *top verifiers* and to the other 100 verifiers as *helper verifiers*.

The information flow is as follows. At the beginning of the round, the top verifiers download the full status report of the previous round, that is, $STATUS_{t-1}$. Since this status consists of a 30MB, the top verifiers can accomplish this task even with a cell phone.

Now, let us discuss the preferred information flow for each of the envisaged 1000 payments of round t. Consider a payment from a (payer) public key P_i to a (payee) public key P_j. Then, since we are envisaging using a tree-hash-and-sign algorithm, (the owner of) P_i retrieves from the cloud the 2KB proving the individual status of P_i and forwards this proof to (the owner of) P_j together with his 100-Byte payment; (the owner of) P_j preferably verifies both pieces of information and forwards them to each of the 10 helper verifiers in a bucket associated to one of the 10 leafs. Otherwise said, the payee selects a bucket B and forwards the relevant payment and individual status to each verifier in B.[6]

Note that the computation involved so far is trivial: (the owner of) P_i generates one digital signature; while (the owner of) P_j verifies one digital signature and computes one hash to select the bucket B. Also the bandwidth is low: (the owner of) P_i downloads 2K bytes from the cloud and forwards 2.1K bytes to P_j; and (the owner of) P_j forwards 2.1K bytes to each of the 10 helper verifiers in bucket B.

[6] All other information flows that convey the essential information should be considered part of the inventive architecture. For instance, the payer may only sends his payment to P_j; while P_j downloads from the cloud the individual status of PK_i, and forwards it and the payment to the helper verifiers of the selected group. Alternatively yet, the payee only forwards the payment to the helper verifiers, who then downloads the individual status of PK_i of the previous round from the clouds. Etc.

In our architecture the bucket B is preferably selected *at random*. In particular, to ensure that lazy payees do not instead always select —say— the first bucket, B can be selected via a given cryptographic hash function H. For instance, (the owner of) P_j may hash the payment (possibly with additional information, such as v_t) and use the last decimal digit of the hashed payment to determine which of the 10 possible buckets of help verifiers should be the selected B. (In this case, a helper verifier in B, receiving and processing the payment information, will also use H to verify that the payment information was correctly sent to the bucket B to which he belongs.)

Since 1000 payments are distributed randomly across 10 buckets, each bucket of verifiers will be selected for approximately 100 payments. Accordingly, each helper verifier must be able to receive 2.1K bytes from approximately 100 users. Therefore, even via a standard cellphone he can receive this data within 1 minute (even being capped at 10 simultaneous connections).

At this point, each helper verifier checks that, for each payment he handles, all the relevant information is correct. (That is, for each payment of \$X from a public key P_i to another public key P_j, he checks that the digital signature of the payment, the digital signature(s) of individual status report of P_i for the previous round, and that the amount X does not exceed the money attributed to P_i in that report.) Then, he preferably compiles all valid payments in a single, preferably ordered, list L, and digitally signs L together with an indication of the round t (e.g., the current time). Finally, he sends to each of the 10 top verifiers, his signed and dated list L, and preferably also the individual prior status of each paying public key.

To receive this information, each top verifier only need to open 100 connections to download the countersigned payments. Again, even using a standard cell phone, this can be done within 1 minute.[7] Each top verifier i then produces the reports $STATUS_t^i$ and PAY_t^i, and posts them on the cloud. The size of both of these reports is about 31 MB and can be uploaded to the cloud within 4 minutes by a cell phone. Additionally, also the signed information received from the helper verifiers could be uploaded to the cloud, so as to keep everyone accountable.

More Efficient Updates. We highlight that its possible to optimize this design by uploading/downloading only self-contained records between the verifiers and the cloud. For instance, suppose at round $t-1$ all verifiers and the cloud hold up-to-date status report $STATUS_{t-1}$. Suppose a top level verifier i wants to efficiently communicate $STATUS_t^i$ to the cloud. Using tree-hash-and-sign mechanism, a hash of every record of the report corresponds to a leaf in the tree; and only the root of the tree needs to be signed by the verifier. There are at most 2000 record changes between $STATUS_{t-1}$ and $STATUS_t$, given 1000 payments. Hence, assuming the cloud knows $STATUS_{t-1}$, to communicate $STATUS_{t-1}^i$, the verifier only needs to communicate the changes in the tree (that is, 2000 new records), and the signature of the new root. Given $STATUS_{t-1}$ and the new records, the cloud can reconstruct the entire tree and obtain the hash of the root. It then can use the signature of this hash obtained from the verifier to reconstruct the full version of $STATUS_t^i$. Using this mechanism, only approximately 210K bytes of data needs to be communicated between the verifiers and the cloud.

In sum, the Urban Instantiation can be run via a cell network with

- *Network Bandwidth and Capacity:* 1 Mega-bit per second (Mb/s), capped by 2 GigaBytes (GB) per month, and

[7]Note, that is may be tempting to store reports produced by the helper verifiers on the cloud and ask the top level verifiers to retrieve it from there. However, this is bad design decision since in this design the cloud may choose to deny an individual's payment by erasing the corresponding reports from the helper verifiers.

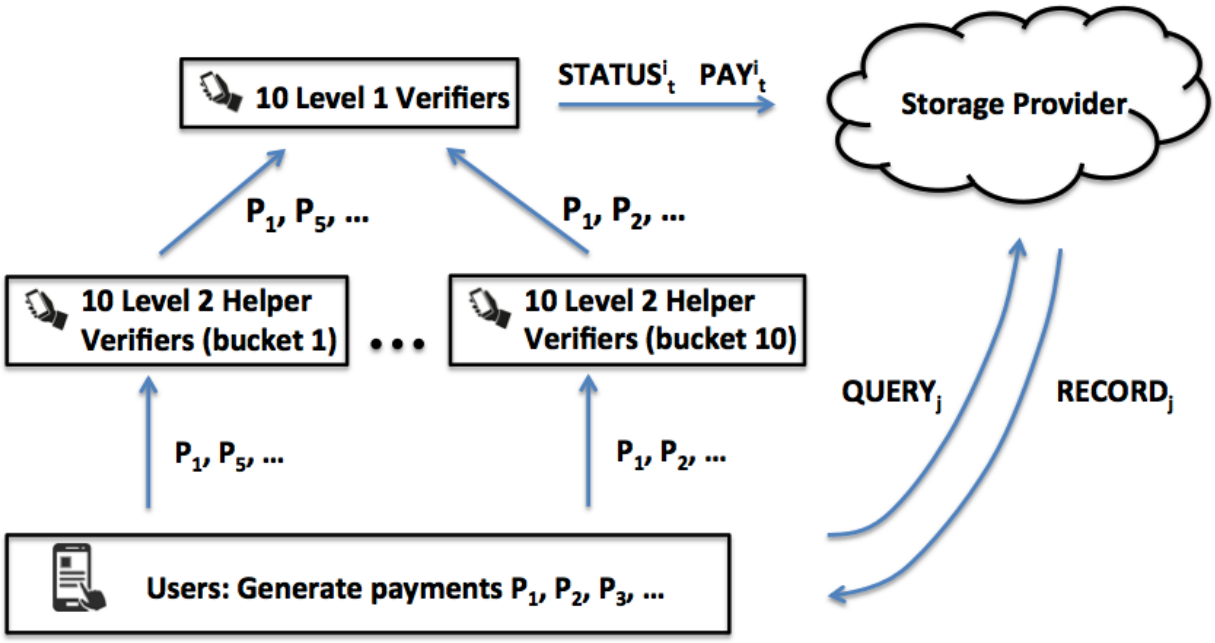


Figure 1: Information Flow in Democoin on Urban Instantiation Example. Using a cryptographic hash function, each user selects a bucket of level 2 helper verifiers to which it forwards the payment. Helper verifiers check the payment, countersign and forward it to the level 1 verifiers. Level 1 verifiers check all payments and forward reports $STATUS_t^i, PAY_t^i$ to a storage provider/facilitator. Any user can then query the storage provider for a short self-contained authenticated record either from the status reports or from the payments reports to update its state information about the system.

- *Connections Capacity:* 10.

Regional Instantiation. A regional instantiation is so defined:

- *Number of Users:* 3,000,000.
- *Number of transaction every 10 minutes:* 10,000.

That is, in the Urban Instantiation, users and transactions numbers are 10 times higher than in the Urban one. We envision running a Regional Instantiation via laptops.

This time the total size of the status report is about 300 MB, that is 10 times bigger than before. Thanks to the tree-hash-and-sign method, however, the size of self-contained report about an individual public key is only 5K bytes. To maintain a good performance, we increase the number of helper verifiers by a factor of 10, so that the total number of verifiers, from 110, now becomes 1010. We partition all verifiers in 101 groups (buckets) of 10, and conceptually assign each bucket to a node of a 3-level 10-degree tree T. (I.e., the root of T has, as before, 10 children (of level 2), but now each of these children has 10 children (of level 3).) The 10 verifiers assigned to the root are again called top verifiers, and all other verifiers are helper verifiers.

There are now 100 buckets on the third level. Each payee (or payer) of a given current payment chooses at random (using a cryptographic hash function) which bucket will handle the payment. Therefore, on average, each level-3 helper verifier only handles 100 payments, which can be downloaded within a minute by a laptop computer. Once verified, these payments are forwarded to a bucket of level-2 helper verifiers (again, chosen at random). Similarly, each level-2 verifier needs to open only 10 connections and download 1000 payments, of size approximately 100 Kilobytes which can be done easily within a minute. After verifying the information received from his children buckets, and after combining them, and signing the combined information, each level-2 helper verifier, sends the proper information to each of the top verifiers, that is, to each verifier in the group associated to the root, the only node of level 1. After verifying, combining, and signing all the received information, that is, after generating $STATUS_t^i$ and PAY_t^i, each top verifier i uploads them to the cloud. This uploading would take about 4 minutes with a standard laptop.

(Again, thanks to the envisaged tree-hash-and sign method, each top verifier i can use the same "More Efficient Update" method, already discussed for the Urban Instantiation case, in order to dramatically cut down the amount of data to be uploaded, and thus the time of the uploading, without increasing at all the trust of the power of the cloud.)

Finally, any player may update its state by querying the storage provider for relevant self-contained new records within a minute. The computational time needed from verifiers is also about a minute. Therefore, the envisioned duration of a round in this instantiation is below 10 minutes, in fact it only takes 8 minutes.

In sum, the Regional Instantiation can be run (within 10 minutes) with

- *Network Bandwidth and Capacity:* 10 Mb/s, capped by 80 GB per month, and
- *Connections Capacity:* 10.

National Instantiation. The National Instantiation is so defined:

- *Number of Users:* 30M.
- *Number of transaction every 10 minutes:* 100K.

In this instantiation, we scale the institutional example by a factor of 100. That is, suppose there are 30 million users and 100 thousand transactions every 10 minutes. The size of the status reports grows to 3 GB, but the size of self-contained authenticated individual record remains small – about 7 KB. We also add a 4th level of 1000 buckets of helper verifiers. Therefore, they are now an additional 10000 verifiers at the $4th$ level, each receiving about 100 payments for verification. All other parameters scale correspondingly and can be easily handled by a standard laptop. Moreover, the entire round can easily be performed within about 20 minutes using efficient-updates mechanism described in the Urban Instantiation (the communication overhead for 100K payments is insignificant, but we have to add extra 10 minutes to account for the computational time to verify these payments).

Even Larger Instantiations Generalizing the architecture discussed above, one can easily handle Continental and Planetary Instantiations, respectively capable of handling 300M and 3B users, and 10× and 100× as many transactions as the National Instantiation. The adoption of the *More Efficient Updating* will be crucial here. The rounds will become longer. But feasibility will still be maintained.

8 Democracy, Fairness, and Security

Democracy. Democoin is a democratic monetary system in that the responsibility of running it resides with the users themselves. For efficiency reasons, however, Democoin is not run by all users *simultaneously.* Rather, at each round, only some users are randomly select to act as verifiers, so as to guarantee the integrity of the system. The verifiers of a given round are rewarded for their effort and availability. Indeed, they stand to be collectively paid 1% of the total money that changes hands in that round. No payment is made by the users to outside parties for running the system. (Except for the payment due to the cloud for providing accessible storage, but this is a negligible amount relative to that traditionally due to "trusted parties" to run a financial system).

Fairness. Better yet, Democoin is fair.[8] A verifier of a given stands to make a lot of money, and, at every round, all users have the same probability of becoming verifiers. Nonetheless, a user who is *never* selected during his lifetime to be a verifier and has *always* paid 1% of his money transfers to other users, may not go to a better life feeling that the system was "fair to him." It may thus be useful to analyze how often a player expects to be a verifier in the three instantiations of Section 7.

Assume that the total 1% reward of a given round is distributed equally among all verifiers (i.e., that top verifiers and helper verifier are treated alike). Then, because the ratio of total number of users and the total number of verifiers is roughly the same in the Urban, Regional, and National Instantiation, in all these instantiations the probability for a user to become a verifier is the same at every round. Moreover, because in the first two instantiations each round consists of 8 minutes, it can be easily seen that in the Urban and Regional Instantiation a player is expected to become a verifier about 22 times a year, that is, just under 2 times a month. Altogether this is a non-trivial frequency. This frequency can be increased by increasing the verifiers-to-users ratio.

Security. Let's now consider the probability of overall security. The system works as envisaged if the majority of the verifiers in every bucket is honest and execute correctly according to the system specifications. Suppose that 90% of the users are honest and hence that a status report (or payment) is valid if and only if 90% of verifiers validate it. (If such a 90% consensus is not reached in a given round, then de facto "no money has changed hands in that round.") In this case, the probability that a randomly selected player is malicious is 0.1, and the probability that a randomly selected pair of players are malicious players is $0.1 * 0.1 = 0.01$. Continuing this calculation, we can derive the probability that 9 or more verifiers, at a given bucket, are chosen to be malicious, which is equals to about 10^{-8}.[9] Now, in Urban Instantiation there are 11 buckets of verifiers. Hence, the probability that one of them is malicious is at most $1.1 * 10^{-7}$. Hence, every 9 million rounds, we expect one bad selection of the verifiers. Given 8 minute rounds, we expect about $65,744$ rounds in a year. Hence, we expect a bad occurrence once every 137 years, which is enough for practical instantiations on the Urban scale.

However, suppose we increase the number of verifiers in each bucket to 50 and suppose 80% of them are honest. Then, the probability that a single bucket of verifiers is malicious (that is, 40

[8] Democoin is also fair, in a difference sense, when the probability of being selected is proportional to the money a user owns, possibly over different public keys.

[9] Given ten selected players in a bucket, the probability that one of them is not-malicious and the rest are malicious is $10 * (0.9 * 0.1^9)$. We also have to account for the case when all players are chosen to be malicious which is 0.1^{10}. Summing up the two probabilities, we obtain that from the selected 10 players, 9 or more are malicious is about 10^{-8}.

or more of the selected verifiers are malicious), is about $1.3 * 10^{-19}$.[10]. Now, given 11 buckets in the Urban Instantiation, the probability that one of them is malicious is at most $1.43 * 10^{-18}$, or once every $7 * 10^{17}$ rounds. Alternatively said, we expect a bad occurrence once every 10 Trillion years. Moreover, in Regional and National Instantiations, this frequency reduces (but remains astronomically high) to approximately every 1 Trillion and 100 Billion Years.

Moreover, note that in order to mount a successful attack, it is not enough that at least 40 of the 50 selected verifiers are malicious, but that these malicious verifiers succeed in coordinating their actions within a round, that is, in a few minutes. (Recall in fact that the set of verifiers of a future round t cannot be predicted long in advance, because it depends on a variable v_t that is totally unpredictable until round t.) Since a round is very short, this coordination may not be practically feasible. Moreover, by slightly increasing the number of verifiers, it is possible to achieve virtually any level of security deemed useful.

9 Tree-Hash-and-Sign Authentication

Tree-hash-and-sign is an effective mechanism to authenticate a large list of records with a single signature, while supporting efficient "local" verification (without the need to download the entire list). This mechanism is used in many existing payment systems, such as Bitcoin. We summarize it in this section for completeness.

Suppose a verifier V wants to tree-hash-and-sign a list of payments $PAY = (P_1, \ldots, P_n)$. (This procedure can be used analogously on any list of records, such as a list of players' accounts information). First, he builds a Merkle tree of the list PAY, inductively starting from the leaves and converging at the root. The leaves of the tree are associated with level 0 and the root of the tree is associated with level $q = \log n$ (for simplicity, we suppose $n = 2^q$ for some q). To compute the Merkle tree, the verifier first hashes the individual payments P_i and associates the hashes to the leaves of the tree:

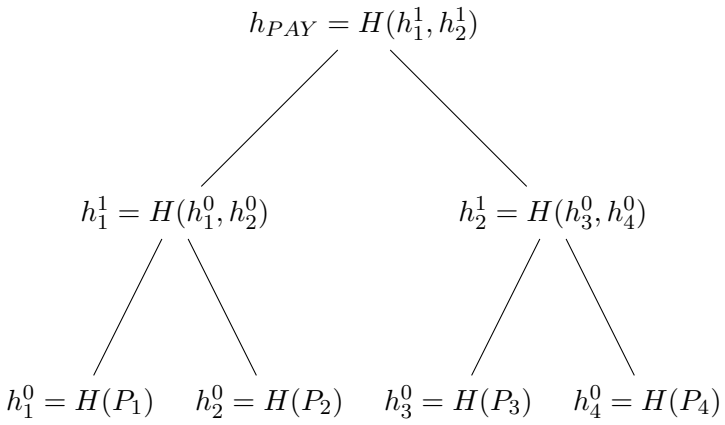
$$h_i^0 = H(P_i)$$

Then, inductively, he computes a hash for a node by hashing two of its children. In particular, to compute a hash h_j^i for a node at level i (in range $1, \ldots, q$), let h_c^{i-1} and $h_{c'}^{i-1}$ be the two hashes for the children of the node. Then,

$$h_j^i = H(h_c^{i-1}, h_{c'}^{i-1}).$$

The hash $h_T = h_1^q$, associated with the root of the tree, is a "commitment" to the entire list of payments PAY. For example, suppose $PAY = (P_1, P_2, P_3, P_4)$, then the Merkle tree the verifier would compute would look like:

[10] This probability p can be derived using formula: $p = \sum_{i=40}^{50} \left(\binom{50}{i} * 0.2^i * 0.8^{50-i} \right)$. Here, 0.2 is the probability of selecting a malicious verifier, and 0.8 is the probability of selecting an honest verifier; and we sum up over all "bad" selections.



To authenticate the list, the verifier may publicize a single digital signature $SIG_V(t; h_{PAY})$, where t denotes the time.

Now, a self-standing V-authenticated record about P_i consists of:

1. the payment P_i itself (and optionally the hash of every node in the path from the leaf corresponding to P_i, $h_i^0 = H(P_i)$, to the root, h_{PAY}), and
2. the hashes of all siblings of the nodes along that path (together with any payments associated with the leaves it downloads) and
3. V's digital signature of the hash of the root, $SIG_V(t, h_{PAY})$.

For instance, in the above example, the self-standing V-authenticated record about the payment P_1 consists of
(1) P_1 (and optionally, h_1^0, h_1^1, h_{PAY}),
(2) the associated sibling hashes h_2^0 and h_2^1 and the payments at the leaves P_1, P_2, and
(3) $SIG_V(t, h_{PAY})$.)

To verify this authenticating record of P_i one can:

1. verify the signatures of the individual payments at the leaves,
2. compute the hash of the root, h_{PAY}, by recomputing, in a bottom-up fashion, the hash of every node along the path to the root

   (In fact for each such node he has already computed the hash of one of its children, c, and has retrieved the hash of the other child, that is, of the sibling of c) and
3. verify the signature of the root: $SIG_V(t; h_{PAY})$.

Again, referring to the above example, the owner would check that P_1 and P_2 are valid payments, whether

$$h_T \stackrel{?}{=} H\Big(H\big(H(P_1), H(P_2)\big), h_2^1\Big)$$

and that $SIG_V(t; h_{PAY})$ is a valid signature.

Efficiency. It is easy to see that the verifier's computation is very efficient. He only needs to evaluate an efficient hash function (for example, SHA-512) to construct the Merkle tree. Hence, the total number of hashes he needs to compute for n payments is $2n - 1$ (corresponding to the total number of nodes in the tree). Since hashing is very efficient, it would take less than a second for a standard computer to produce a Merkle tree for, say, a million payments. Then, the verifier needs to produce a single digital signature authenticating the entire list. For example, using one of the standard Elliptic-Curve Signature Algorithms, the time it would take to produce such a signature would be around 2 milliseconds, and would be around 200 bytes (including in it a large amount of useful information deemed useful).

It is also easy to see that the player needs to download very little information from the verifier and that its computation is efficient. In particular, the player downloads the path consisting of $\log n$ hashes, the sibling $\log n$ hashes (and the two payments at the leaves in clear). Since logarithmic function remains very small even for astronomically large n, the total number of hashes the player needs to download and recompute remains very small. Moreover, it only needs to perform a few (three) signature verification algorithms.

Using a standard Elliptic-Curve Signature Algorithms (used in Bitcoin and other payment systems), we present a summary highlighting the efficiency of this approach in Table 1. It is easy to see that even for astronomically large 1 Billion payments, only (approximately) 31 Kilobytes needs to be downloaded by a player. Since even a cellphone with a weak internet connection can download at a rate of (at least) 1 Megabyte per second, this download rate can easily be handled. Moreover, the verification time remains very small, and is largely dominated by the verification of the signatures.

# of Payments	Path Length	PDownload Size (KB)	PVerify Time (ms)
128	7	8	10
1024	10	11	10
32768	15	16	10
1048576 ($\approx 10^6$)	20	21	11
1073741824 ($\approx 10^9$)	30	31	11

Table 1: Tree-hash-and-sign approximate efficiency evaluation. Above, "Path Length" stands for the length of a path of hashes a player needs to download to authenticate a payment; "PDownload Size" stands for player's total number of Kilobytes he needs to download; "PVerify Time" for player's time (in milliseconds) to verify this record (for a standard cellphone).

References

[1] *Bitcoin Block Chain Info*, https://blockchain.info, Feb 2015.

[2] D. L. Chaum, *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms*, Commun. ACM, Volume 24, Number 2, Pages 84–90, 1981.

[3] *Bitcoin Computation Waste*, http://gizmodo.com/the-worlds-most-powerful-computer-network-is-being-was-504503726, 2013.

[4] S. King, S. Nadal, *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*, 2012.

[5] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.

[6] *Litecoin*, https://litecoin.org/, 2011.