NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF UNDERGRADUATE STUDIES

UNDERGRADUATE THESIS

Efficient blockchains with contributed randomness

Ioannis K. Konstantinou

SUPERVISOR: Aggelos Kiayias, assistant professor

ATHENS

JUN 2016

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΠΡΟΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Efficient blockchains with contributed randomness

Ιωάννης Κ. Κωνσταντίνου

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: Άγγελος Κιαγιάς, επίκουρος καθηγητής

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2016

UNDERGRADUATE THESIS

Efficient blockchains with contributed randomness

Ioannis K. Konstantinou
R.N.: 1115220800057

SUPERVISOR: Aggelos Kiayias, assistant professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Efficient blockchains with contributed randomness

Ιωάννης Κ. Κωνσταντίνου
Α.Μ.: 1115220800057

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: Άγγελος Κιαγιάς, επίκουρος καθηγητής

ABSTRACT

We present a distributed ”proof-of-stake” e-cash system that can tolerate an adversary controlling up to $1/3$ of the stake. We argue that bitcoin's proof-of-work can be replaced to some extent by an unbiased source of randomness and an assumption on the volatility of honest players' majority during each user's offline time. Also we describe a way for players to insert unbiased randomness into the blockchain.

SUBJECT AREA: Cryptography, Distributed Systems

KEYWORDS: e-coin, cryptocurrency, blockchain, proof of stake, unbiased randomness, secret sharing, proof of security, bitcoin

ΠΕΡΙΛΗΨΗ

Παρουσιάζουμε ένα ηλεκτρονικό νόμισμα σε ένα κατανεμημένο “proof of stake” σύστημα που μπορεί να ανεχτεί αντίπαλο που θα κατέχει μέχρι και τα μισά του συνόλου των νομισμάτων. Επιχειρηματολογούμε ότι η ενεργοβόρα “απόδειξη εργασίας (proof of work)” του Bitcoin μπορεί να αντικατασταθεί σε κάποιο βαθμό από μια αμερόληπτη πηγή τυχαιότητας και μια υπόθεση για την μεταβλητότητα της πλειοψηφίας των έντιμων παικτών κατά την διάρκεια της περιόδου που κάποιος παίκτης είναι εκτός δικτύου. Επίσης περιγράφουμε έναν τρόπο ώστε οι παίκτες να εισάγουν τέτοια αμερόληπτη τυχαιότητα στο σύστημα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Κρυπτογραφία, Κατανεμημένα Συστήματα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: ηλεκτρονικό νόμισμα, αμερόληπτη τυχαιότητα, απόδειξη ασφαλείας

ΕΥΧΑΡΙΣΤΙΕΣ

Για την εκπόνηση της παρούσας Πτυχιακής Εργασίας, θα ήθελα να ευχαριστήσω τον επιβλέποντα επίκ. καθ. Άγγελο Κιαγιά, για τη συνεργασία και την πολύτιμη καθοδήγησή του.

CONTENTS

ΠΡΟΛΟΓΟΣ

Η παρούσα εργασία εκπονήθηκε στο εργαστήριο κρυπτογραφίας του επικ. καθ. Άγγελου Κιαγιά. Η εργασία χρηματοδοτήθηκε μερικώς από το ERC πρόγραμμα CODAMODA.

1. INTRODUCTION

In the core of the bitcoin protocol lies its leader election procedure. The leaders for the next block are the ones that have managed to solve the computational puzzle defined by the previous one. The cryptographic properties of the puzzle assure us that there is no better way for solving it than searching the space of possible solutions by brute force. Now imagine that players of the protocol are all the units of computational power available to its users. Then each player has an equal chance of finding the solution and become the leader. This stochastic process picks a leader uniformly from all players. So every real world entity has probability of becoming the leader proportional to the players it controls.

The issue with the described process is that the more valuable becomes a leadership in the real world, more players (a.k.a. computational power) will have incentive to join the election by trying to solve the puzzle. So there exists a lineary analogous relationship between the computational power spend in the election process that ultimetaly secures the blockchain and the total value depicted/secured by the blockchain. Bottom line, the problem is that as the economy supported by the blockchain grows, so does the computational cost to secure it.

As an attempt to counter the above, there exist ”proof-of-stake” protocols. Their difference is that the players are not all units of computational power available, but holders of the currency. If someone wants to enter the group, he has to persuade another to transfer coins to him. This is a less open group, but the good part is that its size does not immediately translate to computational cost. Now what is vital for the protocol is a computationaly light random process that will elect the leader for every step of the protocol uniformly from all players. That part is the core difference between one ”proof-of-stake” protocol and another.

1.1 Related work

There are several attempts to introduce a new cryptocurrency without the computational cost of bitcoin.

”Democoin: A Publicly Verifiable and Jointly Serviced Cryptocurrency” [5] presents the ’Democoin’. A ”proof of stake” cryptocurrency that randomly picks a set of users from the ones referenced in the blockchain to be the leaders for the next round. But for the source of its randomness it relies on a external randomness beacon (e.g. stock market prices).

’Peercoin’ [6] is perhaps the most popular ”proof o stake” coin. Its source of randomness depends on the blockchain for input so an adversary can try and manipulate it to his favor. Also Peercoin uses centrally broadcasted checkpoints several times a day, to ensure consensus on a chain.

”Cryptocurrencies without Proof of Work” [7] lets every player in the sequence of leaders to randomly pick a bit and then combines them all together to elect the next sequence

of leaders. Depending on the combining function, the adversary needs to corrupt certain players in the sequence to bias the result in case the are players inputs are equally divided between 0 and 1.

”math of nxt forging” analyzes the leader election process of NextCoin, another popular ”proof-of-stake” coin. It shows that an adversary can gain significant bias if he manipulates it properly and under certain conditions take over the protocol completely.

”CentrallyBankedCryptocurrencies” [8] does not use proof-of-work but is designed for the case where participating players are a group of banks and are trusted more than the typical person.

All coins above need a form of randomness beacon to make leader election a random process where the result is not known far in the past. In almost all cases the protocols specify a way for the users to collectively produce fresh randomness that is based on their inputs to the blockchain. The problem is that there is always room for the adversary to bias this result if he controls a number of key players. The most simple case being having control of the last player that outputs the value, if he dislikes the output he can choose to absent. And because this result is used to ”randomly” sample the players who will provide the next set of inputs, this process can potentially continue to increase the bias until the adversary completely takes over.

Our contribution is a blockchain protocol that in parallel to building the blockchain, it produces randomness that is unbiased by the adversary and can be used in place of an external randomness beacon.

2. OVERVIEW

In section 5 we describe the 'static shares' protocol. A simple "proof of stake" leader based protocol that executes in synchronous rounds and at every round it aims to extend the longest chain available. Where a trusted entity provides the players with a random mapping from rounds to leading players, before the protocol starts.

In section 6 we give a formal analysis of the protocol and show that it maintains chain-quality and common-prefix properties [1] against byzantine adversary controlling up to $1/2$ of the total stake.

In section 7 we describe a public verfiable secret sharing scheme as presented in [2]. It will be used as building block in section 8.

In section 8 we extend the 'static shares' protocol by allowing stake holders to transfer their stake to other public keys. We also introduce a way for players to collectively insert unbiased randomness into the blockchain. The new randomness prevents the attacker from using stake transfers to launch extra attacks.

In section 9 we show that the new mechanism for randomness is equivalant to the previous case where a trusted party provided the randomness and then no coin transfers were possible. This way the results of section 6 still apply.

3. SETTING

All players start with a common reference string that is generated by a trusted entity and is used to initialize the protocol. The protocols executes in synchronous rounds. All players have an independent trusted way of determining which is the current round at any given moment. At the end of each round a message broadcasted by an honest player is delivered to all honest players.

The adversary can abuse the broadcast primitive and selectively send different messages to different players. An upper limit of 1/2 of all players may be corrupted by the adversary. The adversary is computationally bounded and standard intractability assumptions must hold for the cryptographic primitives that are used.

We also require a signature scheme $sign(x, privKey)$, $verify(y, x, pubKey)$ so that

$$verify(sign(x, privK), x, pubK) = True$$

4. DEFINITIONS

Blockchain

Let as indroduce the blockchain notion. A block is a quadraple of the form

$$B = \langle h, content, round, openings, sgn \rangle$$

where $h \in \{0,1\}^k$, $content \in \{0,1\}^*$ and sgn is a signature of $h \| content \| round$. A blockchain C of length l is a sequence of l blocks $B_1, .., B_l$ so that

$$B_i = \langle Hash(B_{i-1}), content_i, round_i, sgn_i \rangle, \forall i \in [2, l].$$

We define $Head(C)$ the most recent block in C (so if C is of length l this will be B_l). Furthermore, we define $C^{\lceil k}$ to mean the chain C without the last k blocks and $C^{<r}$ to mean the portion of chain C where blocks have roundstamps less than r.

Definition 4.0.1 *(k-Common-Prefix). The **k-common-prefix property** states that at any given round and for any honest players $p1, p2$ having local chains $c1, c2$, it holds that:*

$$max(|c1 - c2|, |c2 - c1|) < k \in \mathbb{N}$$

where chains are viewed as sets of blocks, so $c1 - c2$ means the blocks of $c1$ not included in $c2$.

Definition 4.0.2 *(Chain Quality). The Chain-Quality property with parameters $\mu \in \mathbb{R}$ and $l \in \mathbb{N}$ states that for any player P with chain C, it holds that for any l consequtive blocks of C the ratio of adversarial blocks to honest ones is at most μ/l.*

When we talk about l-chain-quality in the future we will refer to the case of 1/l ratio, fixing the first parameter to 1.

5. THE STATIC SHARES PROTOCOL

5.1 initialization

There are N players, all having a public key pair. They all share an agreement on the N public keys that participate in the protocol, an enumeration of this keys (PK_1,...,PK_N) and a random number $rand \leftarrow [1, N^L]$. They also know two cryptographic hash functions $G : \{0,1\}^* \mapsto [1, N]$ and $H : \{0,1\}^* \mapsto [1, 2^{256}]$. We denote $Leader(r)$ the public key of the leader of round r and define it as $Leader(r) = PK_{G(r\|rand)}$. So essentially there exists consensus on a sequence of L public keys $Leader(1), .., Leader(L)$.

Notice that the random number must be generated after the enumeration of the keys. Else the adverasry could choose an enumeration so that for example he has the leadership in k consequtive rounds, and thus trivially create a fork.

A blockchain C will be considered valid only if $\forall B_i = \langle h, x, round, sgn\rangle : B_i \in C$, $round$ strictly increases on i, $round \leq curentRound$ and
$verify(sgn, H(x\|h\|round), Leader(round)) = true$.

5.2 execution

At each round the players receive all valid competing chains from the network and adopt the longest one. The leader of the current round can then extend upon the longest chain he received. He will include in his newly created block all non-conflicting content he received from his peers. At the end of the round everyone broadcasts his local chain to the others.

Algorithm 1 'static shares' peer loop

1: $C \leftarrow \emptyset$
2: **while** true **do** ▷ for all network rounds
3: $C \leftarrow$ maxvalid($C \cup$ all chains in Receive())
4: $leader \leftarrow$ Leader(r)$= PK_{G(r \| CRS.rand)}$
5: **if** $myPK = leader$ **then**
6: $C \leftarrow$ extend($C, sk, commitment$)
7: **end if**
8: Broadcast(C)
9: $r \leftarrow r + 1$
10: **end while**

11: **function** extend(C, SK)
12: hash $\leftarrow H(head(C))$
13: content $\leftarrow$ all content in Receive()
14: signature $\leftarrow$ sign($H(content, hash, r)$)
15: **return** $C \cup \{\langle r, hash, content, signature\rangle\}$
16: **end function**

6. STATIC SHARES ANALYSIS

We will examine under which conditions the core assurances provided by the protocol - common-prefix and chain-quality - hold.

6.1 common prefix

Lemma 6.1.1 *Any two honest players never produce blocks for the same height (at any chain).*

Proof. If an honest block was produced at height l then at the next round every honest player has a chain of at least l blocks. Also notice that a player never switches to a shorter chain.

We first establish the necessary conditions for the common-prefix property to be broken in rounds s through r.

Proposition 6.1.2 *For k-common-prefix to be broken in $S = [s, r]$, are required to exist M rounds lead by the adversary and H rounds lead by the honest online players such that $M \geq H/2$.*

Let's assume that in the current round k-common-prefix property is broken.

There are at least 2 honest players $p1$ and $p2$ with chains $c1$ and $c2$. Let C be their common prefix. Let $c2 \geq c1$. By definition $c2 - C$ must be at least k blocks long. For $p1$ to have $c1$ means that this is the first round any honest player has any chain greater than $c1$ (else it would be broadcasted). So all $len(c2) - len(c1)$ most recent blocks of $c2$ must be produced by the adversary.

For the heights that both $c1$ and $c2$ have blocks:

Let i be the number of heights for which blocks exist in both chains ($i = len(c1 - c)$). Since honest players produce at most one block for a given height(see Lemma 6.1.1), the adeversary must produced at least one block for each height (either on $c1$ or $c2$).

In total the adversary has produced at least $|c2 - c1| + i = |c2 - C| \geq k$ blocks. Notice that the adversary can produce multiple blocks for the same height and so both blocks of a specific height.

Let $w = |c2 - C| \geq k$, which is the number of blocks required of the adversary. Roundstamps must be stricly increasing over height, so each leadership may be used at most once per chain. At best, the w adverserial blocks can be splitted evenly between the two chains, and so only $w/2$ leaderships may be needed to produce them. Since the same leadership can be used on both chains, for different heights. All this adversarial rounds must be after the round of head(C) and not after the current one.

Let S be the set of rounds between that of head(C) and the current one. The rounds in S lead by honest online players cannot be more than $len(c1 - C) \leq w$ since every honest round produces a block that increases a chain's length by exactly one and is made public. In other case there would be a public chain $c3$ such as $len(c3) > len(c1)$ and $p1$ would not adopt $c1$.

So for k-common-prefix to be broken in S, are required to exist M rounds lead by the adversary such that $M \geq w/2 \geq k/2$ and H rounds lead by the honest online players, such that $H \leq w$.

$$M \geq w/2 \wedge H \leq w \Rightarrow M \geq H/2$$

There can also be an unbounded number of offline rounds (when the respective leader is honest and offline).

6.2 common prefix probabilistic analysis

Given the initialization of the protocol we call $BadEvent$ the event that in any of the L rounds of execution, exist the necessary conditions for the adversary to break the k-common-prefix property.

$$BadEvent = \bigcup_{\forall a,b \in [1,L]: a+k<b} Bad_{S=[a,b]}$$

$Bad_{S=[a,b]}$ means that k-common-prefix can be broken in rounds $S = [a, b]$. Specifically $M_S \geq H_S/2$ and $M_S \geq k/2$.

$$BadEvent = \bigcup_{Of=0}^{L} [OfflineRounds = Of] \bigcap \bigcup_{\forall a,b \in [1,L-Of]: a+k<b} Bad_{S=[a,b]}$$

Constrained only to online rounds, $M_S \geq H_S/2$ becomes $M_S \geq \frac{|S|}{3}$.

$$P[Bad_S] = P[2M > H \wedge M \geq k] < P[M > \frac{|S|}{3}] = P[M > A \cdot |S| \cdot (1 + \frac{1/3 - A}{A})]$$

Notice that $A \cdot |S|$ is the expected value of M. So using Chernoff bound we get:

$$P[Bad_S] = P[M > A \cdot |S| \cdot (1 + \delta)] \leq e^{-\delta^2 A \cdot |S|}$$

where $\delta = \frac{1/3-A}{A}$.

$$P[BadEvent] \leq P[\bigcup_{\forall a,b \in [1,L]: a+k<b} Bad_{S=[a,b]}] = \sum_{|S|=k}^{L} (L - |S| + 1) e^{-\delta^2 A |S|}$$

$$< L \cdot (L - k + 1) \cdot e^{-\delta^2 Ak} < L^2 \cdot e^{-\delta^2 Ak} = e^{2 \ln L - \delta^2 Ak}$$

So for the next L rounds the probability the adversary creates a fork k blocks deep, asymtotically falls exponentially in k.

For an example, let's set $L = e^{20}$ and assume $A = 1/4$. Then $P[BadEvent] = e^{40-k/4} = e^{(1440-k)/4}$. So the probability a fork of $1440 + d$ blocks may form in the next L blocks (rounds), falls exponentially on d. Practically, for an honest player this means that stability of the chain is guaranteed, as there will be no long forks in the future.

6.3 chain quality

For the chain quality property to be subverted, the adversary must own k consequtive blocks in the chain. In any large enough($\geq k$) sequence of rounds, with very high probability the adversary has leadership close to $\frac{1}{3}$ of them and the rest are honest. So without maliciously excluding honest blocks from the prevailing chain, the chain quality will hold.

Now for an honest block to be excluded from the prevailing chain, the adversary must use one of his own in its place. So in a sequence of k rounds where the $\frac{2}{3}$ are honest and produce blocks, half of them may be replaced by using all the adversarial ones. But still there will always be (close to)$\frac{1}{3}$ honest blocks that the adversary cannot match and will end up in the chain.

So at worst case the chain-quality will be 1/2.

7. PUBLICLY VERIFIABLE SECRET SHARING

Our next protocol utilizies the PVSS protocol presented in [Sch99] for its leader election process, so we will briefly describe it.

There is a dealer and n shareholders. The protocol consists of two phases, the sharing and the reconstruction of the secret. For the reconstruction to be succesfull, t out of n shares are required.

Discrete logarithms equality

We will use the protocol by Chaum and Pedersen [CP93] as a subprotocol to prove that $\log_{g_1} h_1 = \log_{g_2} h_2$, for generators $g_1, h_1, g_2, h_2 \in G_q$. We denote this protocol by DLEQ(g_1, h_1, g_2, h_2) and it consists of the folowing steps, where the prover knows α such that $h_1 = g_1^\alpha$ and $h_2 = g_2^\alpha$:

1. The prover sends $\alpha_1 = g_1^w$ and $\alpha_2 = g_2^w$, with $w \in_R \mathbb{Z}_q$.
2. The verifier sends a random challenge $c \in_R \mathbb{Z}_q$
3. The prover responds with $r = w - \alpha c \pmod q$
4. The verifier checks tha $\alpha_1 = g_1^r h_1^c$ and $\alpha_2 = g_2^r h_2^c$

7.1 initialization

We publicly select a group G_q of prime order q, such that computing descrete logarithms in this groups is infeasible. We also independently select generators G, g of G_q, so no party knows the discrete logarithm of g with respect to G. Also every player i publishes a public key of the form $pk_i = G^{sk}$.

7.2 distribution

First the dealer pick a random polynomial p of degree $t-1$ with coeefficients in $\mathbb{Z}_q$:

$$p(x) = \sum_{j=0}^{t-1} \alpha_j x^j$$

and sets his secret to be G^{a_0}. The dealer keeps this polynomial secret but publishes the related commitments $C_j = g^{\alpha_j}$, for $0 \leq j \leq t$. He also publishes the encrypted shares $esh_i = pk_i^{p(i)}$, for $1 \leq i \leq n$. Finally, let $X_i = \Pi_{j=0}^{t-1} C_j^{ij}$. The dealer shows that the encrypted shares are consistent, by producing a proof of knowledge of the unique $p(i), 1 \leq i \leq n$, satisfying:

$$X_i = g^{p(i)} \quad , \quad esh_i = pk_i^{p(i)}$$

The non-interactive proof is the n-fold parallel composition of the protocols for DLEQ(g, X_i, pk_i, esh_i). Aplplying Fiat-Shamir's technique, the challenge c for the protocol is computed as cryptographic hash of all $X_i, esh_i, \alpha_{1_i}, \alpha_{2_i}$, $1 \leq i \leq n$. The proof consists of the common challenge c and the n responses r_i.
Verification of the shares. The verifier computes $X_i = \Pi_{j=0}^{t-1} C_j^{i^j}$ from the C_j values. Using $pk_i, X_i, ehs_i, r_i,\ 1 \leq i \leq n$ and c as input, the verifier computes α_{1i}, α_{2i} as

$$\alpha_{1i} = g^{r_i} X_i^c, \quad \alpha_{2i} = pk_i^{r_i} esh_i^c$$

and checks that the hash of $X_i, esh_i, \alpha_{1i}, \alpha_{2i}, 1 \leq i \leq n$, mathces c.

7.3 reconstruction

Decryption of the shares. Using his private key sk, each participant finds his share $S_i = G^{p(i)}$ from esh_i by computing $S_i = esh_i^{1/sk}$. They publish S_i plus a proof that the value S_i is a correct decryption of esh_i. To this end it suffices to prove knowledge of an α such that $pk_i = G^\alpha$ and $esh_i = S_i^\alpha$, which is accomplished by the non-interactive version of the protocol DLEQ(G, pk_i, S_i, esh_i). *Pooling the shares* Suppose w.l.o.g. that participants P_i produce correct values for S_i, for $i = 1, ..., t$. The secret G^{a_0} is obtained by Lagrange interpolation:

$$\Pi_{i=1}^t S_i^{\lambda_i} = \Pi_{i=1}^t (G^{p(i)})^{\lambda_i} = G^{\sum_{i=1}^t p(i)\lambda_i = G^{p(0)}} = G^{\alpha_0}$$

where $\lambda_i = \Pi_{j \neq i} \frac{j}{j-i}$ is a Lagrange coefficient.

8. THE MOVING SHARES PROTOCOL

We now present a new protocol that extends upon the capabilities of the 'static shares'. As opposed to before that they were static, now the "trust shares" that define potential leaders can be transfered from one public key to another any number of times. But now we need to prevent the attacker from planning ahead and acquiring k shares that provide consequtive leaderships. We introduce a generator of randomness into the blockchain that is unbiased and we use it for the leader election. This way we keep the future leaders unknown to any malicious minority that does not break the k-common-prefix and k-chain-quality properties.

Let as fix k to a value -based on our assessemnt of the adversarial power- so that k-common-prefix property holds with very high($1-\varepsilon$) probability. We now define some new special kinds of $content$:

$transfer\{PK_i, PK\}$ that is signed by PK_i and transfers all its leadership rights to PK, effective after $2k$ blocks. Essentially, every player substitues PK_i with PK in his list of potential leaders, when this content is at least k blocks deep in his local chain.

$commit\{PK, r, poly, enc\_share_1, ..., enc\_share_k, DLEQs, sign\}$ which is the sharing phase of a non-interactive public verifiable secret sharing scheme [2]. The secret is a random value that is shared between the k most recent eligible leaders. So if the $commit$ is included in a block with roundstamp r, then the shares are encrypted for public keys $Leader(r-k), ..., Leader(r-1)$. PK is the public key of the commiter and $sign$ his signature.

Definition 8.0.1 (commitment) *We will now define the exact form of a commitment. Let PK, SK be the public and private keys of the commiter and r the roundstamp of the block it's included in. Also let $holder_i = Leader(r-i, C)$, except of $holder_0 = PK$. Then the commitment will be of the form:*

$$PK, r, g^{\alpha_0}, g^{\alpha_1}, .., g^{\alpha_t}, sgn = sign(SK, H(g^{\alpha_0}, g^{\alpha_1}, .., g^{\alpha_t}))$$

$$esh_i = holder_i^{p(i)}, DLEQ(g, g^{p(i)}, holder_i, esh_i) \forall 0 \leq i \leq k$$

Note that besides the shares, the commiter encrypts α_0 to himself and proves it. So in the future he can decrypt it to G^{α_0} and prove it without needing the shares, thus saving space. Also he does not need to keep any local state beside his private key.

Definition 8.0.2 (assert) *In the algorithms to follow, we make use of an $assert()$ statement. It takes as input a boolean value and if it evaluates to false it immediatly breaks execution of its caller, forcing it to return $false$. So 'assert(false)' is equivelant to 'return false'.*

Definition 8.0.3 (DLEQ) *From now on by DLEQ(g, h_1, G, h_2) we denote a structure containing the corresponding DLEQ proof along with the four input arguments.*

Algorithm 2 verify commitment

function verifyCommitment($commitment, LeaderH$)
  $\langle PK, round, DLEQ(g, g^{p(i)}, holder_i, esh_i) \forall i \in [0,k] \rangle \leftarrow commitment$
  assert($verify(PK, H(g^{\alpha_0}, g^{\alpha_1}, .., g^{\alpha_t}), sgn)$)
  for all $i \in [1,k]$ **do**
    assert($holder_i = LeaderH[r]$)
    verify $DLEQ(g, g^{p(i)}, holder_i, esh_i)$
    assert($g^{p(i)} = \Pi_{j=0}^{t}(g^{a_j})^{i^j}$)
  end for
  verify $DLEQ(g, g^{p(0)}, PK, PK^{p(0)})$
  return true
end function

Algorithm 3 find commitment

function findCommitment($PK, r, C, LeaderH$)
  $C_{work} \leftarrow C^{<r}$ ▷ drop blocks after round $r-1$
  $C_{work} \leftarrow C_{work}^{\lceil 2k}$ ▷ trim last $2k$ blocks
  while $length(C_{work}) > 0$ **do**
    B $\leftarrow$ head(C_{work})
    for all $commitment \in B.commitments$ **do**
      if $PK = commitment.PK \wedge \neg$used$(commitment, r-1, C, LeaderH)$ **then**
        return $commitment$
      end if
    end for
    $C_{work} \leftarrow C_{work}^{\lceil 1}$ ▷ search chain backwards
  end while
  for all $commitment \in CRS.commitments$ **do**
    if $PK = commitment.PK \wedge \neg$used$(commitment, r-1, C, LeaderH)$ **then**
      return $commitment$
    end if
  end for
  return ε
end function
function used($commitment, r, C, LeaderH$)
  if $LeaderH[r-1] = commitment.PK$ **then**
    return $true$
  end if
  if findCommitment($PK, r-1, C, LeaderH$) $\neq commitment$ **then**
    return $false$
  end if
  return used($commitment, PK, r-1, C, LeaderH$)
end function

```
1: function Leader(r, C, R, LeaderH)
2:     if ∃LeaderH[r] then
3:         return LeaderH[r]
4:     end if
5:     i ← 0
6:     while true do
7:         leader ← PK_{G(i‖r‖R[r−1])}
8:         if findCommitment(leader, r, C, LeaderH)≠ ε then
9:             LeaderH[r] ← leader
10:            return leader
11:        end if
12:        i ← i+1
13:    end while
14: end function
```

We also require a new field $open$ to be present in each block header, which is the oppening of a (uniquely specified by the protocol) previous $commit$. It must open the most recent unused $commit$ that is at least $2k$ blocks deep in the chain and was commited by the creator of this block. A $commit$ is considered used if Also if this block has roundstamp $r2$ and the previous one in the chain has roundstamp $r1$, we require that the header includes all the openings that were to be published in rounds $r1$ to $r2$. Basically the $open$ of all skipped roundstamps (provided either by the issuer or the honest shareholders) so there is not the option of opting out and biasing the result.

Given a chain C, we denote R_i the secret randomness on the commitment that was elected to open at round i.The R_i of all the rounds are multiplied together to provide for an unbiased by the adversary source of randomness on the blockchain:

$$rand_r = R_{r-1} \cdot R_{r-2} \cdot R_{r-3} \cdot ...$$

The definition of $Leader()$ changes so that instead of the initial randomness, it uses the collectively produced one. So $Leader(r, C) = PK_{G(i\|r\|rand_r)}$, where r is the round in question, C the chain that is extended and i a counter that is increased until a PK owning a valid commit is output. The i counter is used in order to save silent rounds when a invalid leader is elected. Notice that the final value of i for each round is totally defined by C. In cases were C and i are implied by the context -such as above- we may ommit them.

8.1 initialization

The initialization of the protocol is done by trusted entity. It generates and publishes the group description G_q along with its two generators g, G as required by the PVSS scheme. Also publishes a random element from G_q, an ordered list of players' public keys and a commitment for each one of them. The shareholders for the commitments are chosen uniformly random and independently from the trusted entity.

Algorithm 4 chain validity

1: **function** valid($C, currentRound, R, LeaderH, C_{prev}, k$)
2: assert($head(C).round \leq currentRound$)
3: assert($|C_{prev} - C| < k$)
4: $LeaderH \leftarrow$ []
5: $R \leftarrow$ []
6: reset $PK_1, ..., PK_N$ to CRS values
7: **for** $i = 1$ to $length(C)$ **do**
8: assert(processBlock($B_i \in C, C, R, LeaderH$))
9: **end for**
10: **return** true
11: **end function**

Note: in the bootstraping phase (fist $2k$ blocks) the only commitments that can be opened are in the CRS. So a player cannot have 2 leadership rounds during this phase, because he has no second commitment to open. Meaning the first $2k$ leaders are not truly independent. But it does not hurt our analysis.

8.2 execution

Whenever a players wishes to add new content to the public chain, he broadcasts it. At the start of each round every honest player examines the chains in his network tape and local memory. He picks the longest of them (measured in number of blocks) and discards the others. Let's call this chain C. If he has the private key $mySK$ corresponding to $Leader(currentRound, C)$ then he assembles a new block $B_{new} = \langle h, x, currentRound, sgn, openings \rangle$. Where $h = Hash(Head(C))$,
$sgn = sign(h \| x \| currentRound, mySK)$ and x is a maximal accumulation of known content. Maximal in the sense that no network received content exists that could be added and still be a valid block (because content might impose some rules of its own). By $openings$ we denote the opening of the commitement elected for this round along with any released shares broadcasted for rounds after $head(C).round$. He then appends this new block to the top of his local chain. At the end of every round all honest players broadcast their local chain.

Algorithm 5 processBlock() $\langle r, hash, content, signature, open, skippedOpenings \rangle$

function process($B_i, C, R, LeaderH$)
  $\langle round, hash, content, signature, open, skippedOpenings \rangle \leftarrow B_i$
  assert($round > B_{i-1}.round$)
  for all $r \in (B_{i-1}.round, B_i.round)$ **do** ▷ read openings for skipped rounds
    $com_r \leftarrow$ findCommitment($Leader(r, C, R, LeaderH), r, C, LeaderH$)
    $openings_r \leftarrow$ openings of shares $\in B_i.openings$ for com_r
    assert($|openings_r| = t$)
    $R[r] \leftarrow$ secretReconstruction($openings_r$)$\cdot R[r-1]$
  end for
  $leader \leftarrow Leader(round, C, R)$
  $assert(verify(signature, H(hash, content, r), leader))$
  $R[round] \leftarrow open.secret$

  for all $c \in content.commitments$ **do**
    $assert($verifyCommitment$(c))$
  end for
  for all $t \in content.transfers$ **do**
    $\langle PK_{old}, PK_{new}, r_t, signature_t \rangle \leftarrow t$
    $assert(verify(signature_t, H(PK_{new}, r_t), PK_{old}))$
  end for
  for all $t \in B_{i-2k}.transfers$ **do** ▷ apply transfers that are deep enough
    $\langle PK_{old}, PK_{new}, r_t, signature_t \rangle \leftarrow t$
    $PK_{old} \leftarrow PK_{new}$ ▷ make the substitution on the global list of players
  end for
  return true
end function

Algorithm 6 init

$CRS.players \leftarrow PK_1, ..., PK_N$
$CRS.group \leftarrow G_q$
$CRS.generator \leftarrow g, G$
$CRS.rand \leftarrow^R G_q$
$CRS.commitments \leftarrow \emptyset$
for all $p \in CRS.players$ **do**
  $holders \leftarrow^R CRS.players^k$
  $CRS.commitments \leftarrow CRS.commitments \cup commit(p, holders, secret \in_R Z_q)$
end for

Algorithm 7 maxvalid

```
1: function maxValid(chains, currentRound, R, LeaderH, C, k)
2:     max ← C
3:     for all c ∈ chains do
4:         if valid(c, currentRound, R, LeaderH, C, k) ∧ length(c) > length(max) then
5:             max ← c
6:         end if
7:     end for
8:     valid(max, currentRound, R, LeaderH, c, k)  ▷ reread chain to restore all state (eg. LeaderH)
9:     return max
10: end function
```

Algorithm 8 peer loop

$C \leftarrow \emptyset$
$LeaderH \leftarrow []$ ▷ leadership history for all past rounds
while true **do** ▷ for all network rounds r
  $C \leftarrow$ maxvalid($C \cup$ all chains in Receive(), currentRound, R, LeaderH, C, k)
  $orphanOpenings \leftarrow$ shares found in Receive(), released at rounds after $head(C).round$
  if $head(C).round < r - 1$ **then**
    $R[r-1] \leftarrow$ secretReconstruction(openings for round $r-1$ found in $orphanOpenings$)
    $R[r-1] \leftarrow R[r-2] \cdot R[r-1]$
  end if
  $leader \leftarrow$ Leader($r, C, R, LeaderH$)
  $commitment \leftarrow$ findCommitment($leader, r, C, LeaderH$) ▷ the commit to be opened now
  if $myPK = leader$ **then**
    $C \leftarrow$ extend($C, mySK, commitment$)
  else
    releaseShare($commitment, mySK$) ▷ seek out if i am shareholder for this round
  end if
  Broadcast($C, orphanOpenings$)
  $r \leftarrow r + 1$
end while

function extend(C, SK)
  hash $\leftarrow H(head(C))$
  content $\leftarrow$ all content in Receive()
  c $\leftarrow$ findCommitment(myPK,r,C)
  open $\leftarrow \langle G^{p(0)}, DLEQ(G, myPK, G^{p(0)}, myPK^{p(0)}) \rangle$
  skippedOpenings $\leftarrow$ #threshold shares for each round $\in (head(C).round, r)$ as found in Receive()
  signature $\leftarrow$ sign($H(content, hash, r)$)
  return $C \cup \{\langle r, hash, content, signature, open, skippedOpenings \rangle\}$
end function

function releaseShare($commitment$)
  if $\exists i : commitment.holder_i = myPK$ **then** ▷ i was dealt a share
    $share \leftarrow commitment.esh_i^{mySK}$ ▷ decrypt esh_i to $G^{p(i)}$
    Broadcast(r, i, $DLEQ(G, myPK, share, myPK^{p(i)})$)
  end if
end function

9. MOVING SHARES ANALYSIS

9.1 reduction

The whole analysis for the ”static shares” protocol security, relies solely on the fact that the leader of every round is an independent random variable. We will show that this propery is preserved in the ”moving shares” protocol and so all the previous security arguments for chain-quality and common-prefix still hold.

The system is bootstrapped with a sequnece of leaders satisfying this property and uses this fact to generate more i.i.d leaders.

At any point, the leader of round r is defined by multiplying together k elements of G_q. So to manipulate the outcome the adversary must at some point know all of these values while it's time for him to give his input to the multiplication. But while picking-commiting to this input and until it's permanently burried (k blocks deep) into the blockchain, the protocol specifies that the other $k-1$ elements of the multiplication are kept secret by their owners. So if at least one of these players is honest, the distribution of the outcome over all players is uniform.

Now let's assume these k players have been chosen randomly from a previous same process or the CRS. Since the k parameter has been set accordingly, we are probabilitsticly guaranteed that at least one player will be honest (close to $\frac{2}{3}k$ actually).

So we have established a process, that given a sequence of players chosen uniformly random, outputs a new one chosen the same way. We bootstrap this procedure by feeding it an initial trusted random sequence in the CRS and then let it constantly repeat by consuming its output and appending it to its input. This way as long as enough players participate in the protocol, new i.u.d. leaders are generated.

The adversary has specific actions he can try to abuse to his favor; commiting, opening, transfering and picking wich chain to extend. We saw that while commiting he has not enough info to prefer some group element from another and if he opts out he just loses a leadership opportunity. While opening he is binded by his previous commitment to a spicific value. He can choose not to open, but then the honest majority of the PVSS shareholders will open it instead.

Finally he can try to extend a chain other than the maximum one, in order to erase some blocks he dislikes. But at all times, he cannot erase more than k blocks from the head. Honest commitments are revealed at least after $2k$ blocks -when they are permanent- and before that there is no reason to erase them. Actually since leaders for k blocks in the future are not known, there is no telling of when a newly included commitment will be used. And so even if its secret is known (e.g. commited by the adversary), the other elements of the multiplication are unknown at least until it becomes permanent in the blockchain. At any case, while the commitment is in a block that can be erased, there is not enough info to dislike it. He can try and erase all honest commitments blindly as a denial of service, but chain-quality property assures us that at least one of every k consequtive blocks will be

honest and include all denied commitments. Openings are by design not erasable, since the protocol blocks until openings for all elected commitments are included in the chain.

In conclusion , the adversary has no way -intrinsic to the protocol- to interfere with the process of indepentently electing uniformly distributed random leaders.

Since this procedure is always fed fresh random data from the honest users, its output is unpredictable until all honest players out of k reveal their multiplication inputs. Which means that at worst case where there is just one honest player and he is first in the sequence, the output leader is known by the adversary $k-1$ blocks in advance, but not before. While note that a $transfer$ is applied after $2k$ blocks.

9.2 Long range attacks

Let's define the weight of a subchain as $\frac{length}{currentRound-roundOfFirstBlock}$, so $weight \in [0,1]$ At any point in time, a collusion of past players is able to produce with high probability a chain heavier than the current chain, as long as at the height of the fork they owned a percentage of tokens greater than the weight of the current chain's part they try to overrride. So for example if all the players defined in the genesis block -even if they no longer posses any tokens- decided to produce a new chain, it would have $weight = 1$ and would be the longest possible, able to override any competing chain with at least on silent round. So the longest chain rule is not enough to achieve a persistent blockchain in the long run, when past players may have no incentive to behave honestly any more.

As a counter a measure, we introduce some state information in every player executing the protocol. This state is the longest chain that won during the last online round of the player. When a player receives a longer chain, he will consider it valid only if it excludes at most k blocks of his stored one. So it does not present a fork deeper than k blocks. Else it will just be discarded. This way very deep forks are prevented and players do not need to trust token owners after k blocks from their last transfer. Meaning, a player that transfers all his tokens needs to be trusted only until this transfer becomes permanent (consensus achieved for all honest online players). Then he can no longer reverse it for the set of players that have heard k blocks in a chain verifing it. The remaining offline players who have not yet received this k blocks are still vulnerable.

At any given time a player P_1 has reached a permanent consensus either on a chain burried k blocks deep or on the genesis block. Let's denote this chain as C_C. Block $B_C = Head(C_C)$ is irreversable for P_1. He will later receive a longer chain C_{New} which contains B_C. We denote $Owners(B)$ the set of players that would own tokens if B was the head of the prevailing chain. Player P_1 must trust that $\forall B \in (C_{New} - C_C) \cup \{B_C\}$ there exists honest majority in $Owners(B)$. Where honest means that he was and is still honest until now.

In consequence, for all players to reach consensus, we must trust that a majority of $Owners(B)$ is honest and remains honest until it is burried under k blocks in the local chains of all players.

This is the key difference from the trust assumptions of the Bitcoin protocol. We must trust not only that players act honestly during their leadership round, but that they continue to do so for some time (k blocks) in the future.

To justify this assumption for rational players, we propose to assume that volatility of coin ownership is small. That is, a vast majority of players -coin owners- hold on to their coin balances for long periods of time. And so only a small percentage of coins change hands during a period long enough for all players to come online and hear k blocks from their peers. Giving no incetive to the majority of players to start behaving maliciously.

REFERENCES

[1] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In Advances in Cryptology - EUROCRYPT 2015 Proceedings, Part II, pages 281-310, 2015.

[2] Berry Schoenmakers. ”A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting” In Advances in Cryptology□CRYPTO □99, pp. 148-164, 1999.

[3] D. Chaum and T. P. Pedersen. Wallet databases with observers. In Advances in Cryptology□CRYPTO □92, volume 740 of Lecture Notes in Computer Science, pages 89□105, 1993.

[4] Amos Fiat and Adi Shamir. ”How to Prove Yourself: Practical Solutions to Identification and Signature Problems”. CRYPTO 1986: pp. 186-194, 1986.

[5] Sergey Gorbunov and Silvio Micali. ”Democoin: A Publicly Verifiable and Jointly Serviced Cryptocurrency” , https://eprint.iacr.org/2015/521

[6] Sunny King and Scott Nadal. ”PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake” , https://peercoin.net/assets/paper/peercoin-paper.pdf

[7] Iddo Bentov, Ariel Gabizon, Alex Mizrahi. ”Cryptocurrencies without Proof of Work” , https://arxiv.org/abs/1406.5694v2

[8] George Danezis and Sarah Meiklejohn. ”Centrally Banked Cryptocurrencies” , https://eprint.iacr.org/2015/502.pdf