Private Data Aggregation as a Blockchain Proof-of-Work

December 11, 2018 / v0.11

1 Abstract

We propose a scheme for using the difficulty of the discrete log problem (DLP) over finite groups to simultaneously accomplish two tasks:

1. Collect and aggregate provably-anonymous private data
2. Provide proof-of-work to secure a blockchain

The main motivation for this is to use the same computational power that secures the blockchain network via proofs-of-work to also perform a *useful* task, namely, the provably-anonymous aggregation of private data.

The intent of this writeup is to solicit critical review. This is a novel proposal for securing a blockchain, and as such relies on the accuracy of the claims made to ensure the integrity of blockchain. Unlike hash-based blockchains, this proposal has never seen any use, and has therefore not been subject to any external attack. So, before deploying it, we are looking for strong adversaries to find reasons that the proposal won't work. Primarily we're interested in reasons at the mathematical/theoretical level, as we know there are many practical challenges to putting this into practice. That said, we are interested in all feedback as to how this might fail.

2 General Overview

The system herein described would allow a community/network to:

- collect private data from participants

- with a public record proving which participants' data were collected
- such that any particular person's data is difficult to determine
- yet the community/network can all see and provably validate the accuracy of the aggregated data
- such that invalid contributions/values can be detected (and penalized)

For clarity of discussion, we will use the collection/tabulation of "votes" from participating community members as our motivating example, although other applications are certainly possible. Currently this proposal requires that the data have the property that they can be aggregated together by addition.

3 Technical Overview

The algorithm is based on the difficulty of `DLP` in finite cyclic groups of suitably large size. The network is given a generator g, and participants select a private value α, calculate g^α, and publish this to the network. Miners aggregate many of these values and take the product to get $\prod g^\alpha = g^{\sum \alpha}$, and attempt to solve the `DLP` to determine the logarithm $L = \sum \alpha$.

Since solving `DLP` is hard, this work can be used as a proof-of-work for a blockchain. Miners would be incentivized to solve by mining rewards and, potentially, by additional incentives for data aggregation; since the resources of the entire mining community are working on the problem, hard versions of the `DLP` (i.e., larger group sizes) can be used. Miners would only be rewarded for mining valid blocks, which would require aggregating some minimum number of data values (votes).

The difficulty of solving `DLP` for the aggregate product is nearly[1] the same as solving `DLP` for any single instance; therefore the difficulty of finding out someone's vote is of the same order of difficulty as mining a block, namely, outside the capacity of anyone with less computational power than the combined mining community.

For integration into a blockchain, it is not appropriate for a miner to disclose the actual logarithm L when discovered; otherwise, blocks could be "stolen" and repurposed. Therefore, rather than disclosing L itself, miners publish information that indicates they know L. The security of this

[1] See Section 11.

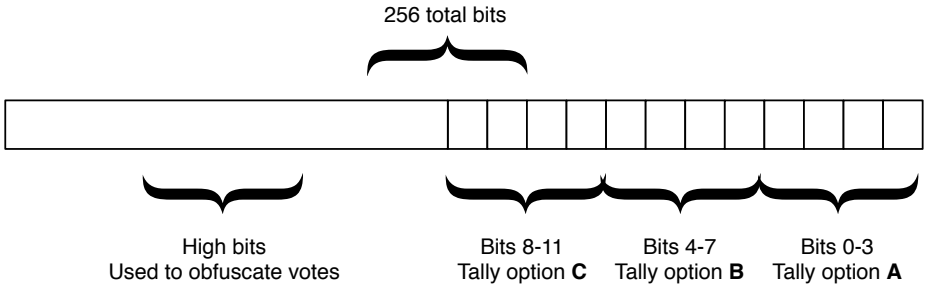

Figure 1: Encoding of data.

approach is based on the difficulty of the "RSA Problem", used in RSA cryptography (as well as the DLP), as described below.

As in traditional blockchains, the proof-of-work also integrates information from previous blocks, making it difficult to create alternative block histories as well as making results from older blocks more secure.

4 Bitwise Encoding

Before digging into details, it will help to introduce some notation and have a general picture of how data are encoded. See Figure 1.

To begin with a simple example, let us assume an aggregation size of $A = 2^b = 16$, represented in $b = 4$ bits; in other words, we assume that fewer than 16 votes will get counted at any one time. (A more realistic size for application is $A = 256$ or $A = 1024$.) This means that we can count all possible tallies for each "candidate" in four bits.

For this example, assume there are three candidates ($|H| = 3$ in the algorithm below). We can use the binary value 0000 0000 0001 (decimal 1) to represent candidate **A**; the value 0000 0001 0000 (decimal 16) to represent **B**; and 0001 0000 0000 (decimal 256) to represent **C**. (These values are chosen because $A^0 = 1, A^1 = 16$ and $A^2 = 256$.) A user U_i's private vote[2] will be represented by a value h_i (in decimal notation), which will be an element of

[2]For this example we are using the term "vote" to denote a single vote for a single candidate, as in the simplest case where each user only gets one vote. However, the approach will easily generalize to less restricted data aggregation.

the set $K = \{1, 16, 256\}$. We will use $\mu = A^{|K|} = 16^3 = 4096$ as a modulus base; note that the binary representation for μ has a single 1 in the lowest of the high bits in Figure 1,

Because there are only three candidates, if a user simply published g^{h_i} it would be easy for an attacker to determine the user's vote; the attacker would simply compute g, g^{16}, and g^{256} and compare to the published value. Instead, the user chooses random bits to insert in the (otherwise empty) high bits of their h_i, resulting in an unpredictable 256-bit α_i. They then publish v_i, given by $v_i = g^{\alpha_i} \bmod n$. In this example, the random high bits allow for $2^{256-12} = 2^{244}$ possible values for each valid vote, effectively requiring an attacker to solve the DLP in order to determine a user's vote.

When a miner aggregates votes, they compute the discrete log of a product of several v_i, and therefore get a sum of α_i. They then simply ignore the high bits to determine the vote tally. Since the aggregation size must be less than A, they can read off the results for candidate **A** in bits 0-3, candidate **B** in bits 4-7, and candidate **C** in bits 8-11.

To be sure the high bits are irrelevant to the tally, we need the sum of all votes in the block (including the randomly-chosen high bits) not to exceed n. If the sum did exceed n, its binary representation would wrap around and affect the low bits where the tally is counted. To avoid this, we require that participants leave the top b high bits empty. In this example that means the top 4 bits are left empty. That leaves 2^{240} possible values for each valid vote, still plenty to require solving the DLP to determine the vote.

The following formalizes these ideas and fleshes out the details required to use this in a blockchain setting. Note that, in fact, a blockchain setting is required for this situation, since to provide the proper security we must have a network with computational power significantly greater than any one entity.

5 Background

Throughout the paper, we use the following notation and terminology:

Definition 1 *A prime number p is called a* **safe prime** *if it can written as $p = 2p_0 + 1$, where p_0 is also prime.*

The numbers p and q will refer to safe primes, with $p = 2p_0 + 1$ and $q = 2q_0 + 1$. Their product $n = pq$ will be used as the base for all modular

arithmetic (unless otherwise specified), as in RSA cryptography. In these cases it is assumed that n is published and p and q are not generally available.

Definition 2 *$\mathbb{Z}_n$ will denote the additive group of integers $\bmod n$, and $\mathbb{Z}_n^*$ the multiplicative group of integers less than n which are relatively prime to n. The* **order** *of an element $x \in \mathbb{Z}_n^*$ is the smallest k such that $x^k = 1$.*

In general, all arithmetic is understood to be $\bmod n$ unless otherwise specified. We will occasionally use $\equiv_n$ to explicitly denote equivalence $\bmod n$.

Definition 3 Euler's totient function *$\phi(n)$ is the order of the multiplicative group $\mathbb{Z}_n^*$, namely, the number of integers less than n which are coprime to n.*

Note that in our case, as in RSA, $n = pq$ with p and q both primes, so $\phi(n) = (p-1)(q-1)$. If p and q are safe primes, then $\phi(n) = 4p_0q_0$. Since the order of an element divides the order of the group, it follows that the order of any element of $\mathbb{Z}_n^*$ divides $\phi(n)$.

Lemma 1 *There exist elements $x \in \mathbb{Z}_n^*$ with order $2p_0q_0$.*

This can be seen by observing that $\mathbb{Z}_n^* \cong \mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_{p_0} \times \mathbb{Z}_{q_0}$, then the element $(0, 1, x, y)$ has order $2p_0q_0$ for any x and y, since $\gcd(2, p_0, q_0) = 1$.

Definition 4 *Let $g \in \mathbb{Z}_n^*$ be an element with order $2p_0q_0$. We will consider $G = \langle g \rangle$, the multiplicative group generated by g modulo n. Note $|G| = 2p_0q_0$.*

Throughout, **difficult** is understood to mean computationally intractable under the correct circumstances; for example, with p, q chosen suitably high. Similarly, **easy** is understood to mean tractable on most modern computers.

The algorithms below are based on the two following claims:

Claim 1 (DLP Claim) *Given $g \in G$ and $h \in G$, it is difficult to determine $a \in \mathbb{N}$ such that $g^a \equiv_n h$, i.e., $a = \log_g h$.*

Claim 2 (Roots Claim) *Given a large exponent a and an integer $h \in \mathbb{Z}_n$, it is difficult to find an x such that $x^a = h$. (This is generally known as the "RSA Problem".) Note that this assumes the private values p, q, and $\phi(n)$ are not available.*

These claims are the foundation of much of current cryptography practice, and are generally believed to be true. Both of these should be understood to be difficult in the general case, with suitably chosen parameters.

Products, inverses, and exponentiations are easy to compute $\bmod n$.

6 Basic Algorithm

To start data collection, someone (TBD) proposes a **topic** $\mathcal{T}$, which consists of:

- The modulus n and generator g as described above.
- $A = 2^b$, the target **aggregation size** (using b bits)
- An integer M such that $MA < |G|$
- $H = \{c_1 \dots c_j : c_i = A^{i-1}\}$, a set of **values** representing the j possible choices

The **modulus base** for the vote is determined by $\mu = A^{|H|}$. The parameter M should be near $|G|/A$; however, we don't want to reveal $|G|$ (as this may provide extra information about the secret p/q values required for RSA exponents to be difficult), so M is chosen to be slightly smaller than $|G|/A$. For this to work properly, we require $M \gg A\mu$; in practice this will generally be satisfied. See the Bitwise Encoding section above for motivation for the choices of these values.

Network participants U_i are interested in submitting their data. They start by creating **private data**:

$$D_i = (h_i, \alpha_i)$$

where:

- $h_i \in H$ is their **vote**
- α_i is a randomly chosen element of the set $\{\alpha \in \mathbb{N} : \alpha \equiv_\mu h_i,\ \alpha < M,\ \alpha \neq h_i\}$

Note that α_i can be chosen by first choosing a random integer $N < M/\mu$, and then computing $\alpha_i = N\mu + h_i$. The set of possible N will be very large as M is much larger than $A\mu$. The requirement that α be smaller than M will ensure that summing fewer than A different α_i's will not result in something larger than $AM = |G|$; i.e., the bits will not wrap around.

Each participant then publishes **public data**:

$$P_i = (v_i, s_i)$$

where:

- $v_i = g^{\alpha_i} \bmod n$
- $s_i = \mathtt{Sig}_{U_i}(v_i)$ is a signature (optional)

Claim 3 *Since* DLP *is difficult in G and the set of possible α's for any given h_i is large, it is difficult for an attacker to determine α_i from v_i.*

6.1 Proto-Mining

First we describe the basic process. (The actual mining process will be slightly more complicated.) A "miner" takes a subset $\{P_1, \ldots, P_m\}$ of the public data, where $A/2 \leq m < A$, and computes:

$$Q \equiv_n \prod_{i=1}^{m} v_i = \prod_{i=1}^{m} g^{\alpha_i} = g^{\sum_{i=1}^{m} \alpha_i}$$

The values v_i are from the public vote data. We require that $m < A$ so that the tallies do not overflow, and we require $m \geq A/2$ so that there is a minimum number of votes included in each block.[3] Then the miner attempts to solve the DLP to obtain the logarithm $L = \log_g Q = \sum_i \alpha_i$. Solving this is difficult and requires the combined resources of the mining network to have a feasible chance at obtaining a solution.

Once a miner solves for L, then anyone can compute:

$$L \bmod \mu = (\sum_{i=1}^{m} \alpha_i) \bmod \mu = \sum_{i=1}^{m} (\alpha_i \bmod \mu) = \sum_{i=1}^{m} h_i$$

i.e., the sum of the values/votes. Note that the middle equality, untrue in general, holds in this case because the $h_i \in H$, and each element of H is less than or equal to $A^{|H|-1}$; this ensures that $\sum h_i < \mu$.

Claim 4 *Knowledge of L confers no information about α_i or h_i for any particular participant U_i.*

In a situation where honest miners are mining honest votes, this seems to be true. We discuss possible attacks related to this in Section 14.

It is easy for anyone with L to confirm the validity of the DLP solution by checking that $g^L \equiv_n \prod_i v_i$.

This is the basic algorithm; variations are required to integrate it into a blockchain, primarily to protect the solution against reuse by other miners.

[3] In practice, the topic will probably specify both the maximum A and some other minimum value, but for the purposes of analysis $A/2$ is suitable.

7 Blockchain Integration

Let the topic $\mathcal{T} = (n, g, A, H)$ be given. We will now discuss blocks, written $B = [\{t_i\}, T, \{P_i\}, I, \pi]$. (When a particular block in the chain needs to be specified, we will add a superscript.)

Definition 5 *A **block** B^j is given by:*

- $\{t_1, \ldots, t_{m'}\}$*, a chosen subset of the available* **transactions**
- T*, the reward transaction for the block*
- $\{P_1, \ldots P_m\}$*, the public value data to be aggregated in this block,* $A/2 \leq m \leq A$
- $I = \texttt{hash}[(t_i), T, (P_i), I^{j-1}]$ *the block* **identifier***, formed by taking the hash of the transaction data, the public value data, and the previous block identifier (the actual checksum function TBD, but for now consider* $\texttt{hash} = \texttt{SHA2} \circ \texttt{SHA2}$*).*
- *A* **proof** π*, which will include the value* $r = \sum_i h_i$*.*

As above, miners attempt to solve for discrete log of $Q \equiv_n \prod_i v_i$; let $L = \log_g(Q)$. Once a miner has found L, they keep this value private. The miner then computes the derived value:

$$r = L \bmod \mu$$

The idea behind this variation is that rather than publish the solution L, miners publish a derived proof π which demonstrates that they have solved for L, but does not allow the solution to be tampered with or stolen by an attacker. The proof must include the low bits of the solution, represented by r, so that the network can verify that the votes were tallied accurately.

Generally speaking, the proof π should be such that the following three claims hold:

Claim 5 *It is difficult to find a suitable proof π without knowing L. ("No fake proofs.")*

Claim 6 *Whether or not L is known, it is difficult to produce a proof π such that $r \neq (\log_g Q) \bmod \mu$. ("No fudging the vote data.")*

Claim 7 *Given a proof π, it is difficult to create a proof π' for an alternative block B' with different checksum I'. ("No stealing block solutions.")*

This is the general formulation that gives the requirements for a suitable proof π to secure the blockchain. The goal is to find a candidate proof that satisfies the claims above; details of the current proposal for the proof π follow in the next section.

8 DLP Proof I: Single Equation

Let the topic and block notation be as before. Once a miner has solved for L and computed r, they can find the unique $k_1 \in \mathbb{N}$ such that:

$$L = k_1\mu + r$$

Because $\mu = A^{|H|}$ and A is a power of 2, multiples of μ will have zeros in the first $b|H|$ bits. This means the values r and k_1 represent, respectively, the low and high bits of the discrete log solution.

Note that the identifier I, derived from the hash of the block, can be construed as an integer (in particular, a positive integer less than 2^{256}). In general, k_1 will be much larger than I; let s be $k_1 \,\texttt{mod}\, I$, and find the unique integer k_2 such that:

$$k_1 = k_2 I + s$$

And then let:

$$R_1 = g^s$$
$$R_2 = g^{k_2}$$

Then the **block proof** π, is given by:

$$\pi = (r, R_1, R_2)$$

This is the public data included in a valid block; the miner keeps L, k_1, k_2, and s private.

A block is considered **valid** if the following equation holds:

$$Q \equiv_n g^r R_1^{\mu} R_2^{I\mu} \tag{1}$$

This is a strong indication that the creator of the proof also knows L, for if a miner created the proof via the methods described, it follows that:

$$g^r R_1^\mu R_2^{I\mu} = g^r g^{s\mu} g^{k_2 I\mu} = g^{(k_2 I+s)\mu+r} = g^{k_1\mu+r} L \equiv_n Q$$

It remains now to demonstrate claims 5-7 above.

Justification of Claim 5 (*No fake proofs*):

To find a fake proof, one must find R_1, R_2, and r satisfying the equation. The chances of picking a valid r is approximately the size of the space of permissible r over the size of the space of possibility (i.e., less than $(A^{|K|})/|G|$, typically $\ll 2^{-256}$); therefore, one would first need to pick a valid r. Then, shifting the g^r to the other side, we get:

$$Qg^{-r} \equiv_n (R_1 R_2^I)^\mu$$

This requires finding a μ-th root, which is difficult due to the Roots Claim.

Justification of Claim 6 (*No fudging the vote data*):

To fudge the vote data, the miner would aim to substitute a different $r' \neq r$ into Equation 1, which requires finding alternate R_1', R_2' such that:

$$Q \equiv_n g^{r'} (R_1')^\mu (R_2')^{I\mu}$$

which also would require finding a μ-th root, which is difficult.

The miner might attempt use the existing Equation 1 to derive a suitable R' by factoring out a $g^{r'}$, giving:

$$Q \equiv_n g^{r'} (g^{r-r'} R_1)^\mu R_2^{I\mu}$$

In order for this to be of the form required, the miner would need to find a μ-th root $x^\mu = g^{r-r'}$, so that $R_1' = R_1 x$ could be provided in the proof. But since $|r - r'| < \mu$ for any two valid r, there's no easy way to factor out a μ in the exponent; therefore finding x is still tantamount to solving for a μ-th root.

Justification of Claim 7 (*No stealing block solutions*):

Suppose an attacker wanted to post an alternative block B' with the reward to alternative going to address T'. This would change the block identifier to I', meaning that an alternative R_1' and R_2' would be required for Equation 1

to hold. Since the attacker does not know L, this is tantamount to the Roots Claim, and therefore difficult.

It also needs to be established that given the block B complete with proof π, it is difficult to determine L. Because R_1 and R_2 are given as a powers of g, it requires DLP and is therefore difficult to determine corresponding values s and k_2. Without these values, r on its own gives insufficient information to determine L.

More analysis is need to validate these claims.

9 DLP Proof II: Separate I/μ equations

An alternate potential proof is described in this Section. More analysis is needed on both proofs to determine which is more secure. The presentation of both options also helps to demonstrate the possibility for the required proof. Both options will be analyzed in parallel.

As in Proof I, let:

$$L = k_1\mu + r$$

And then compute:

$$R_1 = g^{k_1}$$

Note that the identifier I, derived from the hash of the block, can be construed as an integer (in particular, a positive integer less than 2^{256}). As above with base μ, we now break down L in terms of I: let $s = L \bmod I$, and find the unique $k_2 \in \mathbb{N}$ such that:

$$L = k_2 I + s$$

And then let:

$$R_2 = g^{k_2}$$

Then the **block proof** π, is given by:

$$\pi = (r, R_1, s, R_2)$$

This is the public data included in a valid block; the miner keeps L, k_1, and k_2 private.

A block is considered **valid** if the following equations hold:

$$Q \equiv_n R_1^{\mu} g^r \tag{2}$$

$$Q \equiv_n R_2^{I} g^s \tag{3}$$

These are strong indications that the creator of the proof also knows L, for if a miner created the proof via the methods described, it follows that:

$$R_1^{\mu} g^r = g^{k_1 \mu} g^r = g^L \equiv_n Q$$

$$R_2^{I} g^s = g^{k_2 I} g^s = g^L \equiv_n Q$$

It remains now to demonstrate claims 5-7 above.

Justification of Claim 5 (*No fake proofs*):

To find a fake proof, one must find an R_1 and r satisfying the equation. The chances of picking a valid r is approximately the size of the space of permissible r over the size of the space of possibility (i.e., less than $(A^{|K|})/|G|$, typically $\ll 2^{-256}$); therefore, one would first need to pick a valid r. Then, shifting the g^r to the other side, this is equivalent to finding a μ-th root, which is difficult due to the Roots Claim.

After solving this, an attacker must also find an R_2 and s that satisfies equation (2). Fixing s then requires finding an I-th root to solve for R_2, which is difficult; and fixing R_2 requires solving `DLP` to solve for s, also difficult.

There may be some way of simultaneously solving for both R_2 and s, but that seems unlikely.

Justification of Claim 6 (*No fudging the vote data*):

To fudge the vote data, the miner would aim to substitute a different $r' \neq r$ into (1), which requires finding R' such that:

$$g^{r'}(R'_1)^{\mu} \equiv_n Q$$

which would require finding a μ-th root, which is difficult.

The miner might attempt use the existing equation to derive a suitable R' by factoring out a $g^{r'}$, giving:

$$g^{r'}(R_1^{\mu} g^{r-r'}) \equiv_n Q$$

In order for this to be of the form required, the miner would need to find a μ-th root $x^{\mu} = g^{r-r'}$, so that $R'_1 = R_1 x$ could be provided in the proof. But since $|r - r'| < \mu$ for any two valid r, there's no easy way to factor out a μ in the exponent; therefore finding x is still tantamount to solving for a μ-th root.

Justification of Claim 7 (*No stealing block solutions*):

Suppose an attacker wanted to post an alternative block B' with the reward to alternative going to address T'. This would change the block identifier to I', meaning that an alternative R'_2 and s' would be required for the second equation to hold. The analysis above for the first claim above is the same, and comes down to simlutaneously solving for R'_2 and s'.

It also needs to be established that given the block B complete with proof π, it is difficult to determine L. Because R_1 is given as a power of g, it requires `DLP` and is therefore difficult to determine k. Without this value, r on its own gives insufficient information to determine L.

More analysis is need to validate these claims.

10 Sizing and Encoding

The largest `DLP` currently solved over the finite field of integers `mod` p is for an 768-bit prime, which took 6600 core years at 2.2 GHz[4]

More research is needed to determine what an appropriate sizing for a bootstrapped and fully-functional blockchain network would be. To take a quick heuristic: based on the estimate above, the network would require approximately 350 million cores to publish blocks on a 10-minute schedule for

[4] https://en.wikipedia.org/wiki/Discrete_logarithm_records

768-bit primes. So, the network would undoubtedly start with lower primes. On the other hand, 512-bit RSA keys can be factored in a few hours for about $100 on EC2[5], and is a comparable problem to DLP. So best guess would be the network would be initially operating in the 500-600-bit range, increasing from there.

So in a realistic example, we might expect p to be a 600-bit prime, with $A = 2^{10} = 1024$. In a 10-candidate vote ($|H| = 10$), we would then have $H = \{1, 2^{10}, ...2^{90}\}$, and $\mu = 2^{100}$. This would leave 500 bits of randomness for encoding of the α_i, more than enough.

Another possibility is using this system for survey or sentiment data, in which case you'd want to maximize the number of binary values collected. Again using a 600-bit prime, and supposing an aggregation size $A = 2^8 = 256$, and requiring that we leave 256 high bits for sufficient randomization to make `DLP` hard; then we can include up to $\frac{600-256}{8} = 43$ yes/no questions on such a survey.

11 Distribution of DLP/Roots Solutions

The plausibility of the claims of relative difficulty rely on the possible search space of the `DLP` and root problems. Analysis is in progress to determine descriptions of these spaces and the corresponding running times of associated algorithms.

12 Implementation

A proof-of-concept implementation of this algorithm has been implemented in Python, which demonstrates that the mechanics of encoding data, solving the discrete log problem, and reporting results work. Blocks are "mined" by brute-force search for discrete log solutions.

It appears that the best algorithms for finding `DLP` solutions operate in $O(\sqrt{p})$, where p is the largest prime factor of the size of the group; so, p_0 in our case. We could also choose g to have order p_0 instead of $2p_0q_0$; although it's unclear whether this might reveal information about p_0 and/or q_0.

It appears that the best algorithm for finding modular μ-th roots is factorization of the integer n. In this case, $\phi(n)$ can be derived, and a multiplicative

[5]https://github.com/eniac/faas

inverse d such that $d\mu = 1 \bmod \phi(n)$ can be found, as in the decryption exponent in RSA. Then the μ-th root is found by raising to the d-th power.

13 Open Questions

- Note that this relies on the values p and q remaining private, as in the private key material in RSA. Ideally this restriction could be relaxed, but is seems necessary in order to ensure the Roots Claim.
- Nothing in this algorithm should provide any information to make it easy for an attacker to determine p and q. Determining p and q should require the factorization of n (which is difficult), as in standard RSA application.
- Note that μ is, as stated here, not relatively prime to $\phi(n)$. This means that μ-th roots may not exist for a particular $x \in \mathbb{Z}_n^*$. This should make the security claims above more secure, since fudging/stealing requires finding a μ-th root.
- What are the known best algorithms for finding high modular roots? It appears that guess-and-check is best, in which case μ-th roots $\bmod n$ would be much harder than DLP. This appears to be a good outcome for the security of this proposal, since we can tune the difficulty of DLP and have μ-th roots stay consistently harder.
- One question is about the relative difficulty of the DLP and roots problems, particularly for the different parameters; for example, the μ-th root problem is $\bmod n$, whereas the DLP problem is done $\bmod n$, but in a group of size $|G|$ (which is $O(\sqrt{n})$).
- We believe the claims as stated above are strong enough to secure the blockchain. However, they can be made stronger by including a "mask value" α_0 in the vote (see Remarks below). In this case, the security is improved because the set of valid r are not known; the DLP must be solved to determine this. Ideally, we would not require this added restriction (as it complicates the blockchain process), but it could be added if necessary.
- Even if the general problem of DLP is hard to solve, it may be possible that g^{α_i} provides some information about h_i with feasible computational resources. For example, it may be possible that it's easy to determine (given g^{α_i}) if h_i is even or odd; if so, then an attacker could determine whether or not U_i voted for the first candidate. Based on

our understanding of the difficulty of DLP, it seems unlikely that this is possible; but it's certainly an area that will require further analysis. Ideally we could demonstrate that, on the assumption that the DLP difficulty claim holds, g^{α_i} gives you no information about h_i.
- Given the r value from a block, what is the space of possible R/I values? Given an alternate I', perhaps the space of potential R' to search is relatively small, in which case stealing a block is not as hard as the full μ-th root problem?

14 Remarks

- The creator of the topic could choose a random α_0, publish $v_0 = g^{\alpha_0}$ and require miners to solve the DLP for $\alpha_0 + \sum_i \alpha_i$, which would effectively make the vote result private to the topic creator. The creator could then choose whether or not to make the result of the vote public after the fact.
- The use of this α_0 which is private to the topic creator would also mean that it would be even harder for miners to include a fake r in their block, as the small space of acceptable r would not be knowable. Hence, any attempt to submit an invalid r would be detectable (and provable) by the topic creator, and the network could penalize accordingly.
- As mentioned, in addition to v_i, participants could also publish a cryptographic signature s_i proving that the vote is their own. This would allow the network to enforce that values/votes are singly counted, particularly if participants are restricted to known identities, e.g., using an e-identity scheme.
- Given g as chosen above, namely with order $2p_0q_0$, then it's not possible for a μ-th root of g to exist. If it did, then, since μ is a power of 2, we could find an x such that $x^2 = g$. Then $g^{p_0q_0} = h^{2p_0q_0} = 1$, since the maximal order in $\mathbb{Z}_n^*$ is $2p_0q_0$. But that shows $g^{p_0q_0} = 1$, contradicting the claim that it has order $2p_0q_0$.
  This is useful because if there were a μ-th root of g, then a miner could fudge the vote data by adusting R by a factor of the mu-th root of $g^{r'-r}$.
- Note that miners only mine a subset of votes on any particular round, due to the restriction on aggregation size. Thus a topic may last some time, to ensure that all votes are counted. Topics may have times set

before or after which votes cannot be mined. These are all parameters to be tuned by the network.

- The validity of values/votes can be ascertained by examining $r = \sum_i h_i$: since the number of votes is less than A, this sum can uniquely be broken down "base A" as:

$$\sum_i h_i = \sum_{j=0}^{|H|-1} c_j A^j$$

  Uniqueness comes from the fact that there are fewer than A votes cast. The c_j are the counts for each value, and to ensure an accurate count we can check that $\sum_j c_j = m$, the number of votes.
- If the tallies above *do not* add up to m, then at least one participant submitted an invalid vote. This can be addressed by leaving the block as valid (the miner should not be penalized), but not considering any of the votes as counted. The votes are all re-mined, but in different combinations, potentially with some incentive for miners to find bum values. Eventually (within $\log_2 m$ blocks) the bum value will be isolated and the voter penalized.
- The penalization system described above is best effected with requiring stake (coin) to vote: in other words, a participant has to place a certain stake in escrow (have that stake available in their wallet) in order to cast a vote; the escrow is released when the vote is mined in a successfully counted vote.
- The size of the encoding of the primes p and q (roughly $\log_2 p$) will be determined by the required difficulty of the `DLP` in question, tuned to the capacity of the network (in the same way that difficulty is tuned in the bitcoin network to result in a ten-minute block interval). Therefore the solution of the `DLP`, and therefore the security of individuals' data, will be increased with increasing network capability (equivalent of hashrate in bitcoin network).
- This has the problem of not being future-proof, i.e., it is possible that future users will be able to solve `DLP` and therefore determine past votes. One possible solution is to have "vote obfuscation services", which use public-key cryptography to take a set of votes and publish their product, without revealing individual votes. This requires trusting the service; an area for more research is to determine if this is possible to do without requiring third-party trust. (Seems probable

that a solution could be found to this, though also seems unnecessary in the short term.)

15 Version History

- v0.11: Addressed issue with Claim 7. In v0.10, an attacker given a block B with proof $\pi = (R, r)$ could create an alternative block B' with alternative hash I', and use $R' = Rg^{(I'-I)}$. Then:

$$g^r(R')^\mu = g^r R^\mu g^{(I'-I)\mu} = Qg^{I\mu}g^{(I'-I)\mu} = Qg^{I'\mu}$$

  Therefore the proof $\pi' = (R', r)$ is a valid proof for block B', which demonstrates stealing a block solution.
- v0.10: Distribution of aggregated DLP solutions, analysis of comparable difficulty of single and aggregated cases.
- v0.9: First full proposal.