General

Online voting is only beginning to make headway in modern society. It might seem strange that the acceptance of this method of voting has been so slow, but if you take a closer look at the issues involved, the reason will become clear. Online or remote voting is certainly convenient, but it also opens up vast opportunities for spoofing the results. A compromised voting protocol can lead to large-scale changes to the choices made by voters. This means that online voting imposes extremely stringent requirements on the security of every aspect of voting. We believe that the blockchain technology is the missing link in the architecture of a viable online voting system. Below we discuss this in more detail.

Blockchain

Whenever we discuss the blockchain technology, we usually begin with Bitcoin. The reason for that is quite simple: everybody is familiar with the term Bitcoin to some degree or other. Bitcoin operates on the basis of a distributed database of transactions — the so-called blockchain. In even simpler terms, a blockchain is a set of protocols and technologies for distributed storage of transaction data, while Bitcoin is a crypto currency that utilizes such properties of blockchains as decentralization and anonymity to achieve greater stability and earn more trust from users. The blockchain is structured in a way that network participants automatically reach a consensus on the contents of the blockchain records, while guaranteeing the invariability of records and the overall network security. Let me illustrate this using an example: suppose you need to transfer funds using a blockchain protocol. Transactions happen among a certain number of network participants among whom trust is not required. There is also no need for the involvement of a particular centralized trusted authority (such as a bank) that would guarantee the validity of transactions.

How Does it Work?

How can a consensus be reached in a distributed network where participants do not trust one another? The secret is quite simple: a blockchain is not merely a technical implementation of the transaction transfer protocol; rather, it is a kind of ecosystem at the juncture of technologies and economic principles. In other words, a blockchain loses a number of its properties in the absence of economic principles. Let us begin by looking at the blockchain structure from a technical perspective. As its name suggests, it is a chain of blocks:
These blocks and their chain are generated according to specific rules and synchronized among all network participants.
In turn, each block consists of a header and a list of transactions. The block header in turn includes: its own hash, the hash of the previous block, the hash of transactions, and additional service information.
The hash of any block consists of the hash of transactions. As a result, all information about the completed transmission of data is stored, and it cannot be spoofed without replacing the entire block. In this way a block becomes a kind of container for a set of transactions.
Meanwhile, the presence of the hash of the previous block allows arranging blocks into a chain and provides protection against block spoofing.

Let us look at the rules according to which a block is generated. **Here are a few essential principles that guarantee a transparent and decentralized network:**
1. All network users have access to the complete copy of the blockchain and store (or have the capability to store it on their systems;
2. Any network user can generate blocks;
3. Users are rewarded financially for generating blocks;
4. A computing problem has to be solved to generate a block;

Now let us examine this in more detail. The first and second items on the list are more or less clear. We should only mention that storing the entire database copy is not obligatory in modern versions of blockchains. Regarding the third item: users are rewarded for generating a block, which creates an economic incentive to attract additional users who will generate new blocks. This process of creating blocks is known as mining and the person creating them is called a miner. We will come across this term many times subsequently. The fourth item is the most interesting of them all. What kind of computing problem is that? This is where the Proof of Work principle comes into play. Incidentally, it deserves a whole separate chapter.

The problem can vary depending on the blockchain implementation. As a general rule, it involves computing a certain hash in a way that allows solving the problem only using the brute force method.

**Proof of Work (PoW)**
Since Bitcoin is the most popular crypto currency, let us examine the operating principle of the blockchain as exemplified by this currency. Imagine that you have a set of transactions that are not yet included in the blockchain but are only candidates for inclusion. Miners receive these transactions and enter a competition of sorts for who can generate a specific block sooner than others and receive a reward. As I mentioned earlier, generating a block requires solving a certain computing problem. The problem involves generating a hash (the SHA-256 function). It should not be just any kind of hash; its numerical value has to be less than a certain number N. The result of SHA-256 is unpredictable, which is why the right hash can be chosen only using a brute force search. This usually requires a large number of recalculations that use up CPU/GPU resources of the mining computer. Once the block hash satisfying the rules has been found, the miner generates a block and sends it to other network participants for verification. If the block has been generated correctly, it is included in the chain and the miner receives a reward. Note that the rate at which blocks are generated does not change despite the growing network capacity (number of participants). The speed is balanced by complicating or simplifying the computing problem (by increasing or reducing the number N).

**Why is all this needed?** The network is distributed, and sometimes a parallel branch of blocks can form because miners were either unaware of one another or acted wrongfully on purpose (e.g., to perform the transaction in one chain and then spoof it in another). How should the only valid branch be selected? A rule applies whereby the main branch is the one where more CPU/GPU resources have been expended on generating this branch, i.e. essentially the branch containing more blocks. So if a miner starts a separate branch of blocks with its own history of transactions (either accidentally or intentionally), this branch will be declared dead sooner or later and discarded by the network. Transactions in this branch are canceled.

A separate branch can become the main branch only if it is longer than the main branch (black); this can happen if more CPU/GPU resources are expended on generating the branch compared to the other branch. This is in turn possible when somebody controls 51% of the network's computing power.

This was a description of the most basic blockchain operating principles. Now let us move on to online voting and discuss its difficulties and nuances.

### What Are the Difficulties?

**1. Anonymity and Control of the Voting Process.**

Remote electronic voting imposes stringent security requirements on the voting process, because, in serious elections, the temptation is too great and the risk of large-scale manipulation is simply too high. This is why such voting cannot be carried out in black-box mode – the process should be clear and transparent for all participants. It should always be possible to monitor the process and easily verify its results.

**2. Coerced/Influenced Voting and the Ability to Verify That Your Vote Has Been Counted.**

How do you protect people from external pressure or the temptation to sell their votes? Importantly, we still want to have some kind of mechanism to verify that a vote has been counted – and if such a mechanism is in place, it can be used for trading in votes.
3. Transparency and Unavailability of Intermediate Results

Again, keeping in mind the transparency and distributed mechanisms provided by blockchain, we face the issue of intermediate voting results being available.

These are only a few examples. Note that in the process of working on possible threats, sometimes answers to these threats come up which themselves create new threats on a different plane. Which produces a set of threat vectors and answers to these threat vectors with complicated dependencies, eventually shaping the requirements for the system’s behavior. And of course, not all these threats are covered by the blockchain technology alone.

This is why we needed some sort of superstructure on top of the blockchain, which would enable the relevant requirements to be satisfied to a certain degree.

Superstructure on Top of the Blockchain

We will start by listing the most serious problems and tasks that need to be addressed:

- Excessive transparency of the blockchain. If voting results are recorded in the blockchain in unencrypted form, they are distributed among all observers and intermediate results of the election become available, which is against the law and does not make sense.

- Anonymity – each user should be confident that his or her vote is anonymous and the system does not know how that user voted.

- We need to preserve transparency when counting votes and taking each vote into account.

- To preserve trust in the system, voters need a mechanism for verifying that their votes have been counted. At the same time, all other properties of the system should also be implemented.

As a result, we have a situation in which some of the system requirements are mutually exclusive. Obviously, a blockchain with a voting system on top of it is not sufficient – a set of algorithms is needed to satisfy the requirements discussed above.

Algorithm Overview

Below we discuss the main voting stages.

Stage 1. Creating a Vote

Voting deployment → Decentralized key generation → Encryption
At this stage, we need to create a vote, input answer choices and define voting access criteria. The most important thing at this stage is the selection of so-called trusted representatives. These are candidates in the election (or rather their representatives) or independent observers. What are their functions from our algorithm's viewpoint? They perform the following roles:

1. Form blockchain blocks and sign them using their personal keys.
2. Encrypt the blockchain contents to hide intermediate election results.

These two functions are in fact in no way dependent on each other, but we decided to present them together to make it easier for end users to understand how the product works.

**Starting the vote and forming the blocks**

All trusted representatives receive a mining/observer application from us, create their own personal keys with which they will subsequently sign their blocks, deploy the application on servers or desktops and connect to our blockchain. Naturally, all operations are performed automatically. Now, they can receive voter transactions, assemble them into blocks and record them in the blockchain.

**Encrypting the blockchain.** We want to encrypt the voting results recorded in the blockchain. We need a key to do this. However, we cannot use a single key, because:

- it defeats the purpose of having a distributed system
- the key holder is potentially vulnerable: the single key may be stolen, enabling the voting results to be accessed ahead of time, or it may be lost altogether, in which case the voting results will be lost, as well
- the key holder can access the intermediate results

Secret sharing schemes do not have the shortcomings described above. Here are the properties of such schemes that are important to us:

- Each trusted representative has only part of the key, so no one can decrypt the data alone
- To decrypt the data, we do not need all parts of the key but a subset of them, to guard against situations where one of the representatives has disappeared. We use threshold encryption \((n,k)\), where \(n>k>1\), and we need \(k\) parts of the key to decrypt the data. The parameters \(n\) and \(k\) are defined in the process of setting up the system.

Eventually, we chose Shamir’s Secret Sharing scheme.

The scheme is based on the following idea:

You need \(k\) points to interpolate a polynomial of degree \(k-1\). For example, it takes two points to define a straight line, three points to define a parabola, etc. It is impossible to interpolate a polynomial if you have fewer points.

If we want to share a secret among \(n\) people in such a way that it can only be recovered by \(k\) people \((k \leq n)\), we “hide” the secret in the formula of a polynomial of degree \(k-1\). That is, we generate a random polynomial of degree \(k-1\) with our secret being the constant term of the polynomial. Values of the polynomial in points from 1 to \(n\) are sent to all trusted representatives so that each value is sent to one representative. These values can be used to verify that the
secret was created correctly. The polynomial and the original secret can only be recovered using k points. The number of different points of the polynomial is not limited.

Step-by-step description:

At the threshold scheme’s initialization stage, each trusted representative \( P_i \) generates his or her private key \( s_i \). Let us label the system’s common key as

\[
s = \sum s_i.
\]

Trusted representative \( P_i \) generates a random polynomial of degree \( k-1 \),

\[
f_i(z) = \sum_{j=0..k-1} f_{ij} z^j, \text{ where } f_{io} = s_{io} = s_i
\]

after which trusted representative \( P_i \) sends to each trusted representative \( P_j \) (including himself) the polynomial’s value

\[
s_{ij} = f_i(z) \text{ in the point } z=j, \quad j=1..n, \quad i=1..n
\]

Additionally, the following is published:

\[
F_{ij} = g_{ij}^s
\]

where \( f_{ij} \) are the constants of polynomial \( P_i \), and the following check is performed:

\[
g_{ij}^s = \prod_{l=0..k-1} F_{ij}^{i^l},
\]

to prove that \( P_i \) actually calculated \( s_{ij} \) using the polynomial \( f_i(z) \). Additionally, the private key that has been calculated is checked for matching the published parts of the key, i.e.,

\[
g_{i}^s = \prod_{j=1..n} g_{ij}^s = \prod_{j=1..n} (\prod_{l=0..k-1} F_{ij}^{i^l}).
\]

A check is also performed to make sure that these parts match the public key that has been calculated:

\[
\prod_{i=1..n} g_{i}^s = \prod_{i=1..n} F_{i0} = h.
\]

If these checks are passed, then each trusted representative confirms that the public key is correct by signing it.

Stage 2. The Voting Process

At this stage, the task is to receive a vote from a voter and record it in the blockchain. In the process of doing this, the following requirements must be fulfilled:

- Ensure voter anonymity
- Provide protection against trash votes
• Provide protection against vote trafficking and voter coercion
• Enable voters to check that their votes have been recorded in the blockchain.

Here is a step-by-step description of the process:

A. First of all, there is the issue of user identification and each user’s right to vote. This is a major issue, which is resolved differently depending on the segment. For example:

• In a general election, a voter must be a citizen of the country, older than 18 and without a criminal record. In this case, we need to provide integration with user identification systems used in the relevant countries.

• Another example: voting in the OpenStack community (software for creating and managing public and private cloud services) is available to those who uploaded code to a certain GitHub repository during a certain period of time.

B. A user has logged on to the system and we see that he has the right to vote. Now, the voting client application (iOS, Android) generates a private key and a public key that will be used to sign the voting transaction (we will refer to it as a ballot).

C. The user casts his ballot in the client application.

D. The application encrypts the ballot using the common key generated at the vote initialization stage, signs the ballot with the user’s key and sends it to be recorded in the blockchain. Signing the ballot with the user’s key will make it possible to verify that the ballot was delivered to the blockchain intact and that it was saved there successfully.

E. Upon receiving a ballot that has been cast, the voting system verifies that it contains one of the possible choice options, thereby blocking attempts to send trash/spoiled ballots (this will be subsequently used when counting votes). How does this verification work? Below we provide an example of the simplest case of zero-knowledge proof.

In the example above, possession of knowledge of alpha is being proved. Essentially, the above example is an interactive proof, since the Verifier party sends challenge c to the Prover party. However, this scheme can be rendered non-interactive by sending a hash, e.g., of a and b, as “c” when sending parameters a, b, r to the Verifier party.
This principle can be implemented using various mathematical apparatus. We selected the ElGamal system for its properties that are discussed below.

A small remark. The algorithm described here, which is used to encrypt the entire blockchain, ensures anonymity, provided that decryption is not performed at the end. Results of the vote can be counted without decrypting data using the homomorphic properties of ElGamal cryptosystem. More on this below.

It is worth adding that there is a threat of vote trafficking / coerced voting (e.g., when management at an enterprise demands that employees provide screenshots of their ballots). We suggest countering this threat by enabling voters to change their votes without limitation. This will help to break the economic connection with vote traffickers, since a voter can change his or her ballot after selling it to the trafficker. In our case, the issue is resolved as follows: a voter can cast the ballot any number of times, but when counting votes the system will count only the last ballot cast by that user.

Stage 3. Voting Results

Counting encrypted ballots

Now the voting is over and we have an encrypted blockchain. The task is to count the ballots and produce the voting results.

A brief digression. A prime number, starting with 2, is assigned to each voting choice: \( Z_f = \{2, 3, 5, 7, 11, \ldots \} \).

Essentially, when submitting their ballots, voters send these prime numbers, encrypted with the public key, to the voting system.

It is also important to note that the ElGamal system is multiplicatively homomorphic:

\[
E(m_1) \cdot E(m_2) = E(m_1 \cdot m_2)
\]

This means that encrypted message \( m_1 \) multiplied by encrypted message \( m_2 \) is equal to the result of encrypting the product of \( m_1 \) and \( m_2 \).

This enables us to multiply encrypted data without decrypting it. Consequently, the voting results can be saved in the following form:

\[
\prod E(m_i) = E(\prod m_i), \text{ where } i = 1..n, \text{ where } n \text{ is the number of ballots cast}
\]
By performing multiplication on encrypted data we get the product of all votes cast, also in encrypted form.

Decrypting the results

As you remember, we used threshold encryption (Shamir’s Secret Sharing scheme). Any k participants, who know the coordinates of k different points of a polynomial, can reconstruct the polynomial and all of its constants, including the last one, which is the shared secret. Essentially, the task in hand is to solve a system of equations to reconstruct the original polynomial and its constant term. This type of problem is often solved using the Lagrange interpolating polynomial – an approach that we adopted, as well. The underlying idea is simple: this is a polynomial of the lowest degree which assumes the given values in the given set of points, i.e., which interpolates the function by known points. Below we provide some more details on this.

To decrypt the data, we start by identifying the set of trusted representatives, $P_i$, that have parts of the keys and will participate in recovering the original text.

Next, we introduce the set $\pi \subseteq \{1,...n\}, |\pi|=k$.

To recover the original text, we calculate the modified key, $a_i$.

since $s = \sum_{l=1}^{n} f(l(0))$

and $f(z) = \sum_{i=0}^{k-1} f_i \cdot z^i = \sum_{i=1}^{k-1} n_k \sum_{l=1}^{n} n_k s_l (z-j)/(i-j),\$

$s=\sum_{j=1}^{n} \sum_{l=1}^{n} n_k \sum_{l=1}^{n} n_k s_l (z-j)/(i-j)\rightarrow$

$s=\sum_{j=1}^{n} \sum_{l=1}^{n} n_k (\sum_{j=1}^{n} n_k s_l (z-j)/(i-j))$

let us define $l_i = \prod_{j=1}^{k-1} (z-j)/(i-j),\quad a_i = \sum_{j=1}^{n} n_k s_l \rightarrow$

$s=\sum_{j=1}^{n} n_k (a_i l_i)$,

where $a_i$ is the modified key of representative $P_i$, $l_i$ is a multiplier in the Lagrangian function, which depends on node numbers and the total number of nodes in $\pi \subseteq \{1,...n\}$.

To decrypt the voting results, we need to apply keys $a_i$ of representatives $P_i$ sequentially.

Counting the votes

The result is something that looks like this:

$3^{k_1} \cdot 5^{k_2} \cdot 7^{k_3} = N$

where $k_1$, $k_2$, $k_3$ represent the number of people who have voted for each of the candidates and $N$ is the number that we have decrypted. Now that we know the prime numbers used (3, 5, 7) and $N$, we need to factor the equation and obtain $k_1$, $k_2$, $k_3$.

Let us consider a simple case in which we have the following equation:

$\alpha^x = \beta \pmod{M}$

How do we find $x$ if we know $\alpha$, $\beta$ and $M$? The easiest method would be to multiply $\alpha$ by itself until we get $\beta$, since what we are dealing with here is a residue group modulo $M$ and the order
(number of elements) of the group is equal to M, which means that we will need to perform a maximum of M multiplications to find $\beta$ using this kind of enumeration:

This sort of enumeration can be accelerated using Shanks’ algorithm, which is also known as the baby-step giant-step algorithm. The algorithm is based on the following idea:

First, for some $k$ we calculate $\alpha^k, \alpha^{2k}, \alpha^{3k}$, etc. These are the so-called giant steps:

Next, at the second stage of the algorithm, we calculate the products $\beta \cdot \alpha, \beta \cdot \alpha^2, \beta \cdot \alpha^3$, etc. These are the baby steps:

The above diagrams demonstrate that at some point $\beta \cdot \alpha^t$ becomes equal to some $y_m$, which was computed in the previous step. Thus,

$$\alpha^x \cdot \alpha^t = y_m = \alpha^{mk}$$

from which it follows that

$$x = mk - t$$

Careful readers will have noticed that this describes the algorithm for finding one $x$, while we have many unknowns, since there are many choices in an election. This is why the algorithm was slightly modified to turn it into a multidimensional Shanks’ algorithm. A value table (giant steps) was developed for products of $\alpha$ in the following form:
\[
\alpha_1^{i_1 \cdot m} \cdot \alpha_2^{i_2 \cdot m} \cdot \alpha_k^{i_k \cdot m} = \beta =>
\alpha_1^{i_1 \cdot m} \cdot \alpha_2^{i_2 \cdot m} \cdot \alpha_k^{i_k \cdot m} = \beta \cdot \alpha_1^{i_1 \cdot m} \cdot \alpha_2^{i_2 \cdot m} \cdot \alpha_k^{i_k \cdot m}
\]

for \(i_h = 1..m, h=1..k\)

And the baby step in Shanks’ algorithm was performed as follows: for all \(j_h = 1..m, h=1..k\), check whether the number \(\beta \cdot \alpha_1^{i_1 \cdot m} \cdot \alpha_2^{i_2 \cdot m} \cdot \alpha_k^{i_k \cdot m}\) is found in the table. If the number is found, the result is \((i_1 \cdot m-j_1, i_2 \cdot m-j_2, \ldots, i_k \cdot m-j_k)\). The principle is basically the same as above, except that we are working with the product of voter choices from the start.

**Vote counting performance**

Shanks’ algorithm accelerates computation, but the issue of performance in large-scale elections remains open. We carried out a number of one-core computation experiments (to get a feel for the magnitude of things) and the results were as follows:

<table>
<thead>
<tr>
<th>Number of voters</th>
<th>Number of choices</th>
<th>Computing time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>4</td>
<td>0.220 sec</td>
</tr>
<tr>
<td>1,000,000</td>
<td>4</td>
<td>1692 sec (python)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>681 sec (cython)</td>
</tr>
</tbody>
</table>

It can be assumed that, with good optimization and using all processor cores, it would take a reasonable amount of time to process 1,000,000 votes using one computer with 5 choices or 100,000 votes with 6 choices.

It is also important to note that the vote and the subsequent count can be sharded, i.e., assembled into metablocks (in the form of separate blockchains), e.g., 10,000 votes to each metablock. After this, the factoring can be performed separately for each block and the results can then be added together. This will enable the system to be scaled to any size.

**Synergy**

Above, we described a voting protocol. Most of the approaches and techniques described are by no means new – for example, Shanks’ algorithm was introduced in 1972 and the ElGamal scheme in 1985.

So what is so special about the system? It is probably worth mentioning that we used the Ethereum blockchain, an implementation that is often called blockchain 2.0. It is notable for implementing smart contracts, which are essentially scripts, code whose execution is decentralized – in the sense that the correctness of its execution is verified by the network’s participants. In our case, it will be verified by the so-called trusted representatives (observers, candidates’ representatives, etc.).
It is in smart contracts that the voting protocol described above will be implemented. Ballot verification, vote counting etc. will be performed in a decentralized and trusted manner. There will be no black box where ballots are sent to produce a result of the vote arrived at using methods that are not transparent to observers. In fact, observers will, essentially, be an integral part of the system. In other words, the mathematical algorithms that enable us to achieve anonymity, hide intermediate results, perform calculations on encrypted data, as well as implementing these algorithms in the smart contract environment, is that synergy which we believe will provide the much-needed breakthrough in online voting systems.