Possible evolutions of the voting system in Tezos

Véronique Cortier, Pierrick Gaudry, Stéphane Glondu

Université de Lorraine, CNRS, Inria. Nancy, France

January 2021

This is a report written during a collaboration between Nomadic Labs and the authors. The first goal was to add the possibility to keep the votes secret. The general context of Tezos and in particular its fully decentralized nature raises a few questions. In this document we discuss them and propose technical solutions.

Contents

1 General context and current voting system

1.1 The Tezos ecosystem

The Tezos crytocurrency (XTZ) has been released in 2018. One of its features is that the Tezos protocol includes the possibility to amend itself. The amendment process involves a voting procedure that is the topic of the present study.

The money unit for the cryptocurrency is the Tez (about 1 to 3 euros in 2020). Tezos being based on the proof-of-stake paradigm, owning Tez gives rights and responsibility. The stake unit is the roll, which corresponds to 8000 Tez. An entity who owns at least one roll can participate to the construction of a new block in the chain. There is a lottery system that elects a baker for each cycle. This baker will be in charge of creating a new block, and others (the validators) will check and sign this new block as a valid one. An economic protocol, based on safety deposit / reward / punishment, ensures that everyone has a strong incentive to behave honestly.

Additionally, there is a notion of delegation, so that the entities who own less than a roll can have their money participating to the baking process and get some of the reward. A delegate is an entity that will receive delegations from other owners and the rights / responsibility associated to them.

1.2 The four phases of protocol amendment

Currently, the roll is also the unit of rights used for the process of protocol amendment. All the bakers (entities who own at least one roll, maybe thanks to delegation) can propose amendments and vote.

There are 4 phases, each one taking 3 weeks.

1. **Proposal phase.** Each baker can propose up to 20 amendments, during the whole phase. An amendment is designated by the hash of the reference code for this amended protocol. Each baker can also endorse an amendment proposed by someone else.

   The amendment that receives the largest number of endorsements wins this phase and, if this number is above 5%, it is selected for the 3 next phases.

2. **Testing vote.** During this phase, there is a vote of the bakers to decide whether the amendment should be tested.

3. **Testing phase.** There is no vote during this phase where the amendment is tested. A fork is started with the new version in order to detect potential problems.

4. **Promotion vote.** This is the final vote for adopting the new protocol. If the result is positive, this triggers a mechanism on the nodes to switch synchronously to the new version.

For the testing and the promotion votes, there is a single question, and the answer can be “yes”, “no” or “pass”, the last one being a synonym of “blank vote”. Due to its special role in the ecosystem, the Tezos Foundation who owns many Tez always votes “pass”. A supermajority of 80% is required to validate a choice. There is also a participation quorum that must be reached. The value of this quorum is adaptive. If quorum or supermajority is not met, then a new cycle is started, with a first phase where alternate amendments can be proposed.

In the present situation, all the votes are public: the bakers put their (signed) choice in cleartext in the chain. Also, the keys used to sign the votes are the same as the ones used for baking.

1.3 Objectives for a new voting system

There is a list of features that we might want to have in a new voting scheme:

- **Vote secrecy.** Although there are some situations where it is considered important that the votes are public (for instance, the Foundation wants it to be public and verifiable that it voted "pass"), there are also cases where voters might be more comfortable with their opinion being secret.
- **Allow non-baker to vote.** The delegation process used for baking is not necessarily the same as the one for voting. It is not because one trusts some entity to bake on one's behalf, that one trusts it for voting on an amendment of the protocol (that could for instance give advantages to delegates).
- **Resistance to coercion.** One of the reasons for which we require vote secrecy is to avoid coercion and vote buying. However, in Tezos, vote buying is a somewhat built-in feature: it suffices to buy more Tez to get more voting rights. Therefore, in the rest of the document we will not try to resist to coercion and vote buying. We remark however that some solutions are present in the literature, for instance the JCJ/Civitas voting system [5]. Adapting these techniques to the case of Tezos where votes are weighted is not immediate. While it seems feasible, further research would be required to design such a scheme and prove it secure.

Of course, the voting scheme must be compliant with the philosophy of cryptocurrencies, and therefore the process must be **decentralized**, with no central authority in charge for instance of holding the election decryption keys, and **verifiable** by everyone.

The blockchain itself should contain enough information to perform this verification. This does not necessarily means that all the data related to the votes must be inside the blocks. Just like the amendments themselves are not in the blockchain, it might be possible to have some auxiliary data off-chain, as soon as appropriate signatures on the hash of the relevant data is included in the chain.

An additional technical constraint specific to the Tezos blockchain is the notion of context, that should contain all what is needed by the nodes to validate a new element. It is important to keep this context not too large. Section 3.6 is dedicated to this aspect.

2 Setup of the election key

Almost inevitably, an electronic voting system that wants to preserve vote secrecy will require encryption, and therefore a key generation phase. Due to the decentralized nature, this is not as simple as asking some entity to create a key and to send their shares to several parties.

We will think in the asymmetric encryption setting, where a public encryption key is used by the voters to encrypt their choices, and the corresponding decryption key is used at the end to compute the result of the election.

2.1 Choosing the trustees

From now, we will call *trustees* the entities that are in charge of holding the decryption key. Their role is to protect the secret of the votes (only). Even if they collude, all they should be able to do is to learn the votes of the voters, they should not be able to change the result of the election.

The natural candidates for playing the role of the trustees are the bakers. Indeed, what is asked from a trustee is

- Some computing power that is available at pre-defined periods of time;
- Some interest in the general good health of the cryptocurrency;
- Independence from other trustees.

The computing power required for playing the role of a trustee will of course depend on the voting scheme that is implemented. But this should be smaller than what is required to be a baker. Being sure to be available at the right period of time is probably the most difficult part of this aspect, but again this is something that bakers are used to deal with.

The interest in the general functioning of the cryptocurrency can and must be reinforced with a reward / punishment mechanism, just like for baking.

As for the independence of the trustees, it means that voters should be confident that among them, a vast majority are sufficiently honest not to collude with the others in order to break vote secrecy. In more concrete terms, the decryption scheme should ensure that a coalition of less than a threshold of the trustees won't gain any information about the votes, while more than this threshold is required to compute the final result. For example, we could enforce that any coalition of less than 80% of the trustees can not break privacy; choosing the precise value is out of the scope of this study.

We therefore suggest that the trustees are chosen with a process similar to the one used for choosing the bakers. A (publicly verifiable) lottery can be run for the forthcoming election cycle, thus designating a set of bakers to act as trustees. Perhaps, there should be a way for a baker to decline acting as a trustee if it feels that it might not be available at the time of the election. The economical incentive should be appropriately tuned for this to be rare. A large enough number of trustees is required. The precise number is hard to tell without taking into account the current Tezos ecosystem.

A minor difficulty is that the lottery will select an account with a probability that is proportional to the amount of rolls they own. This is necessary to avoid any strategy of splitting / merging accounts. A side-effect is that a large entity who owns many rolls possibly shared on several accounts might be selected twice. Therefore the number of selected trustees and the decryption threshold must be slightly increased accordingly. Furthermore, it should be made possible for entities who got selected twice to act as only one trustee in the key generation phase, for performance reasons.

2.2 Distributed key generation

We propose to use the distributed key generation (DKG) protocol present in Belenios. In most of the other e-voting protocols deployed in practice, there is a central authority that plays an important role, and this does not exist in Tezos. This DKG protocol is described in [2]. The specification [4] and the implementation [1] of Belenios provide an example of how it can be put in practice. The requirements are the following:

- There is a public cyclic group G in which the discrete logarithm problem is supposed to be difficult. A generator g of G is also publicly known. Typical choices are multiplicative groups of a finite field, or the group of points of an elliptic curve over a finite field.

- There is a public board where data can be written by the trustees, so that the process is verifiable. At the end, the encryption key is published on this board. In the Tezos case, the blockchain provides a natural place to implement this board.

- The trustees can authenticate themselves. Again, in Tezos this is the case: a trustee is a roll owner, and has the baking key that can be used as a base of authentication (a sub-key can be derived, if using the main baking key is not appropriate).

- The trustees have private communication channels between them. If this is not already the case, this can be implemented based on the two previous elements: each trustee can publish a (signed) public encryption key on the blockchain, and others can use this to send messages in- or off-chain to this trustee in a private way.

The DKG itself relies on the Pedersen scheme applied to ElGamal. The general idea is to use Shamir's secret sharing scheme based on polynomial interpolation. To make it completely distributed,

each trustee generates its own polynomial, and the sum of all of them is the polynomial that is used for the secret sharing. On top of that, all the polynomial manipulations are done "in the exponent", in the context of the group G where the discrete logarithm is difficult.

The DKG goes through two steps:

1. Each trustee i generates a random polynomial f_i of degree t (the threshold for decryption), and sends commitments to the coefficients on the public board. The trustee number i also sends securely to each trustee number j its polynomial evaluated at j; we call this value $s_{i,j} = f_i(j)$.

2. From the received data, after all relevant verifications, each trustee can construct its secret decryption key, and sends to the board a proof of knowledge of it as an acknowledgement.

The encryption key that will be used to encrypt votes can be computed from the data published during the first step; this is $g^{\sum f_i(0)}$. From the voter's point of view, this is a traditional ElGamal public key in the group G.

We remark that this DKG protocol is of quadratic nature. Let n be the number of trustees and $t+1$ be the number of them required to be able to decrypt the result. Then, during step 1, each trustee produces public data that consists of $t+1$ group elements, and sends privately one scalar to each other trustee. Therefore, in total we are talking about $O(nt)$ group elements to be stored in the public board, and $O(n^2)$ scalars to be sent privately between trustees. Step 2 is very light in comparison.

The decryption phase is simple: each trustee uses its secret decryption key to produce a partial decryption that is sent to the public board. From $t+1$ of them, it is possible to compute the full decryption.

The security guarantee is that if at most t trustees are dishonest, then the secret is preserved, and if at most $n-t-1$ trustees are dishonest, the decryption will happen. Furthermore, there are zero-knowledge proofs added at relevant places so that independently of the trustees being honest or not, if a result is published, then it is correct.

2.3 Size and cost estimates

We assume that the group G is a 256-bit elliptic curve, in order to ensure 128-bit security. In such a setting, with point-compression (*i.e.* using only one bit for the y-coordinate), it takes 32 bytes to represent an element of G (maybe one more byte, depending on the exact curve that is chosen), and 32 bytes to represent an element of the scalar group $\mathbb{Z}_q$. The public commitments of the trustees consists in $n(t+1)$ group elements, so they take $32n(t+1)$ bytes. We consider that the additional n signatures required from the trustees (one for each trustee) require data that is an order of magnitude smaller and do not count them.

As for the data privately sent between the trustees (the $s_{i,j}$), they consists in n^2 elements of $\mathbb{Z}_q$, since there are n of them sent by each trustee. This makes $32n^2$ bytes of data payload. Since it must be encrypted, and since the encryption must be provably decryptable (see below the discussion about accountability), we estimate that there is a factor of 4 that is lost here (this factor will depend on the exact implementation choices). Therefore, we count $128n^2$ bytes of data for the private sending of the $s_{i,j}$'s.

For instance, if $n = 32$ and $t = 28$, this gives 29 kB of data for the public commitments, and 128 kB of data for the private communication of the $s_{i,j}$'s.

The rest of the data is linear in the number of trustees and negligible compared to the quadratic parts that we have just listed.

As for computational costs, during the DKG, each trustee will make a number of exponentiations in the group that is linear in n. At the very end of the DKG, however, the trustees must verify that the final public key correspond to the whole set of commitments, and this requires $n(t+1)$ exponentiations. It might actually be a good idea if the blockchain itself performs this sanity check.

The decryption step is an order of magnitude lighter both in terms of size of the data and in terms of exponentiations.

2.4 Unreliable trustees

The original Pedersen scheme and its variant described in [2] is slightly more complicated than the one deployed in Belenios. During the DKG, if some trustees do not follow the protocol, this will be detected by the honest trustees at Step 2. In Belenios, we consider that such a failure will be rare, so that we can resume from Step 1, excluding the trustees for which the verification failed and the trustees who failed to provide their contribution. In fact, as long as the number of verifications that succeed is larger than the threshold, it is possible to continue the protocol, with a slight modification. This might make sense to implement this robustness feature of the DKG in the Tezos context.

It is even possible to make trustees accountable for their failure: if a trustee T_i detects that the share $s_{j,i}$ received (privately) from T_j does not satisfy the required properties, she can emit a complaint and publish $s_{j,i}$. To make this accountable, we need that the shares $s_{j,i}$ are posted (signed by T_j and encrypted with the public key of T_i) on the blockchain. Then T_i can produce a proof of correct decryption when revealing $s_{j,i}$ so that anyone can check the correctness of the decryption of $s_{j,i}$ and therefore that it does not satisfy the required properties. This way, T_j can be disqualified from the set of trustees, and punished. Note that the protocol can still continue unless the number of qualified trustees is smaller than the threshold. Otherwise, the protocol needs to restart. So probably the punishment of the disqualified trustees should at least reimburse the cost of restarting the election. In fact, an additional parameter could be added, controlling the number of qualified trustees required to open the election, in order to keep a margin with respect to the decryption threshold, in case some of them lose their keys or refuse to use it during the tally phase.

2.5 A hierarchical Pedersen variant

If the number of trustees required to have confidence is high, then maybe the quadratic number of messages to be exchanged during the DKG can be too large to be reasonably stored in the blockchain. A first solution is to store them off-chain, and put only signed, hashed version of them in the blockchain.

Another approach is to use a hierarchical version of the above algorithm. The idea would be to divide the trustees into disjoint subsets, and to run the DKG in each of the subset independently. Then all the resulting encryption keys could be multiplied together, thus providing a single encryption key. In order to decrypt, the DKG decryption would be run in each subset, leading to decryption factors that could then be multiplied together to get the result. The success of the procedure would require that the threshold of participating trustees is reached in all the subsets.

This leads to different compromises in terms of security, robustness and efficiency.

2.6 Key management

The signing keys for the voters can be the genuine secret key associated to the account holding the Tez, or can be derived from it. This is the same for the encryption and signing keys used by the trustees during the DKG: it could be an already existing PKI or can be derived from the key associated to the account.

A more complicated issue is the decryption key of the election that is shared among the trustees. On one hand, it is important to provide the trustees (*i.e.* the bakers) with convenient tools to store it securely, because if too many lose their shares, it will be impossible to decrypt the result. On the other hand, it is also important to offer the possibility to completely delete the decryption key once the election is finished, so that the secret of the vote is maintained beyond reasonable doubt. There is therefore a trade-off to find between usability for the bakers, and long-term security.

Two extreme possibilities are the following.

- The decryption key share could be stored on-chain, encrypted with a key that is derived in a deterministic manner from the main key associated to the account. In terms of usability, this is the best, because the risk of forgetting the decryption key disappears.

- The decryption key share could be stored on a removable media (a USB-stick, for instance), that is put in a safe until the end of the election. Apart from this media, it exists only in a non-swappable segment of the memory of the machine during the operations. Once the election is finished, the data is securely erased from the media. In terms of long-term security, this is the best.

We do not decide between those two, and there might actually be some better trade-off. In a sense the choice might be better left to the baker, since it is not clear that the answer is the same for all.

The question of what to do if the trustee has voluntarily or not made public her decryption key share is left open. Slashing the trustee is of course a possibility but might not be necessary.

3 A weighted Helios/Belenios-like approach

In the academic literature there are three main cryptographic strategies to ensure privacy when computing the results from the encrypted ballots.

- *Verifiable mixnets*. In this setting, the trustees will, in turn, shuffle the ballots in a verifiable way. Once the shuffling is done, if at least one of the trustees keep their shuffling permutation secret, there is no link between the original and the shuffled ballots. Therefore, they can be decrypted individually, and the result can be computed. In our case, due to the weights that often uniquely identify the voters, this would compromise privacy.

- *Homomorphic encryption*. In this setting, the ballots are aggregated using the homomorphic property of the ElGamal encryption scheme: it is possible to get a ciphertext containing the sum of the votes for one candidate, without having to decipher the individual ballots. Only the aggregated ballots are decrypted by the trustees. This is the option that we are going to further explore in what follows. Again, due to weights, some privacy leaks are unavoidable in this setting. This will be discussed below.

- *Multi-party computation*. This is still more an academic than a practical solution, because it requires heavy communications and computations between the trustees. In this setting however, one can decide exactly which information is computed from the encrypted ballots. For instance, it is possible to output (in a verifiable way) only the winner, without any further information. It would be also possible to leak a bit more information, for instance whether the winner won with more than 90%, or between 80% and 90%, etc. Exploring further this MPC approach would require a dedicated work, in order to estimate and optimize the amount of communications and computations that would be required.

We assume from now that we use homomorphic encryption and that the result is not only the winner, but also the number of votes obtained by each option.

3.1 Privacy issues with weighted votes

Elections with weighted votes offer a bit less vote privacy than elections where all voters have the same weight. The first obvious bias is that voters with more important weight have more influence on the result of the election, hence we can learn more about their vote from the result.

Let's consider the list of bakers that can be found at `https://api.tzkt.io/v1/delegates`, that was retrieved on 2020-12-21.

The total amount of rolls was 85800 and only 63169 if we remove the rolls from the Tezos Foundation who always votes "pass". The table below lists the weight of the 10 biggest owners of Tez (including Tez obtained by delegation, but not the Foundation). This corresponds to their minimal relative weight in an election since some owners may not participate in the election (and then the relative weight of the remaining voters increases).

Baker	rolls	(minimal) part in an election, in %
Coinbase Baker	10283	11.98
Kraken Baker	5876	6.85
Binance Baker	5302	6.18
Cryptium Labs	2431	2.83
PayTezos	1954	2.28
Stake.fish	1775	2.07
Everstake	1720	2.00
P2P Validator	1451	1.69
Everstake Legacy	1330	1.55
Coinone Baker	1041	1.21

Then, let's have a look at the election for Carthage 2.0 `https://tzstats.com/election/16`. This proposal has obtained 99.61% of "yes". Assuming that the distribution of rolls was the same as above at the time of the vote (which is of course not the case, but this is for the sake of illustration), we could immediately deduce that all the owners listed in the table, if they participated, voted "yes". Since the "no" has obtained only 117 rolls, we even deduce the votes of the 71 biggest owners. Even in a more disputed election like the one of Babylon 2.0 `https://tzstats.com/election/13`, with 84.53 % of "yes" and 5,846 rolls for the "no", we deduce that Coinbase and Kraken can't have voted for "no" (again, in a fictitious world where the above distribution would have been the same as above at the time of the vote).

These considerations are independent of the voting system. The only way to mitigate such a leak of privacy would be to use MPC techniques to reveal less information than the exact number of voices for "yes", or, in the extreme case, to reveal only one bit of information, namely whether or not there were enough "yes" according to the elections rules.

Another concern is that weighted vote can in some cases also compromise the privacy of voters of small weight. The reason is that the result provides the exact number of rolls obtained for "yes", "no", and "pass". Since the weight of a voter is an integer, it may be possible to deduce the vote of some voters.

In more details, consider n voters where the voter i has a weight w_i (w_i is an integer). Each voter votes δ_i with $\delta_i = 1$ if the voter votes yes, $\delta_i = 0$ if she votes no. It is easy to extend to the case of more voting options (e.g. pass). The result of the election is an integer r representing the number of yes.

$$r = \sum_{i=1}^{n} \delta_i w_i.$$

Now, assume that for some value r and some voter i, the set of solutions $(\delta_1, \ldots, \delta_n)$ to the previous equation is such that, for any solution, $\delta_i = 1$. Then we can deduce that voter i must have voted "yes". Hence, to ensure vote privacy, ideally, it should be the case that the distribution of weights guarantees that whatever the result r is, for any voter i, she might have voted "yes" or "no". More formally, it should be the case that:

$$\forall r \forall i \forall \delta_i. \quad \exists \{\delta_j\}_{j \in \{1..n\} \setminus \{i\}} \text{ such that } r = \sum_{i=1}^{n} \delta_i w_i.$$

Note that actually, this formula is always false since if $r = 0$ then for sure any voter i must have voted 0 (that is "no"). Similarly, as seen already, when r is small (or large), the vote of voters with a high weight is leaked. Hence a more subtle definition needs to be considered to measure the loss of privacy potentially induced by the weights. One may also want to consider:

- whether it can be learned that many voters voted the same way, without knowing their vote (like in the Babylon 2.0 example above);
- the quantity of possible solutions. For example, if there is only one solution to explain that Alice voted yes while all the other solutions indicate that Alice voted no, we may consider the first case to be unlikely. We could also take into account the expected distribution of votes. This can be captured by different notions of entropy that quantify vote secrecy.

Therefore, we need to explore whether the typical weight distribution in Tezos allows for this type of attack against vote secrecy. A very preliminary study of the weights indicates a large number of very small weights (79 accounts with 1 roll, 40 accounts with 2 rolls, 27 accounts with 3 rolls). These small weights should make it possible to hide the differences between the other accounts and allow a large number of solutions. Hence our first conclusion is that weighted elections in Tezos should still reasonably preserves vote privacy, up to the previously mentioned limitations (for voters with high weight).

However, we would like to emphasize that this is a very approximate study. If Tezos wishes to implement weighted vote, a more comprehensive study should be developed, using the concept of entropy to make sure that the only loss of privacy is due to the weight of the very large Tez owners. Further study should also consider the distribution of the weights of those who actually take part in the elections (here our study implicitly assume everyone voted).

3.2 A voting protocol for weighted vote

We describe here the Belenios voting protocol, adapted to weighted vote and to the Tezos context, where the notion of delegation comes into play. Note that the Helios protocol (from which Belenios is derived) has already been used for weighted vote [3]. In the case of Tezos, there is a PKI where each voter has a signing key, which makes the transition to Belenios natural and easy.

3.2.1 Voter list derived from public data

Each voter i has a public key denoted pk_i; this is essentially her `tz1`, `tz2` or `tz3` in Tezos' jargon. The weight of her vote is w_i, for example the number of her rolls (rounded to integer), but we could allow a finer grained unit for voting rights. The correspondence table (pk_i, w_i) can be extracted from the blockchain and is therefore public.

Some voters may have delegated their Tez to others. This delegation is publicly written in the blockchain. We note $i \to j$ if i delegates to j. By default, the delegator vote is assumed to be the vote of its delegate. However, the voter i retains the ability to vote if she wishes. In this case, her delegation will not be counted, her personal vote is counted instead.

3.2.2 Voting phase

We use ElGamal encryption like in Belenios. Let g be the group generator, and h the public key of the election constructed during the DKG. The encryption of a vote $v \in \{0, 1\}$ is the ballot $b_i = c_i, \pi_i, \sigma_i$ with c_i the ElGamal encryption of v:

$$c_i = (g^r, h^r g^v),$$

where r is a random value.

Note that compared to standard ElGamal, we encrypt g^v instead of v in order to benefit from a homomorphic encryption. Hence decryption requires to compute a discrete logarithm, which is fine provided that the sum of the (weighted) votes remains of reasonable size (see Section 3.3).

The remaining of the ballot is composed as follows:

- π_i is a zero-knowledge proof that the vote is valid (either 0 or 1);
- σ_i is the signature of c_i, π_i with the secret key associated to pk_i.

The full details are written in the Belenios specification (apart from the signatures which are handled in a different way). In particular, if voters have more than 2 voting options (typically "yes", "no", and "pass"), then a vote is actually a vector $(v_1, \ldots, v_k)$ and the ballot is formed of a sequence of ElGamal encryptions together with a zero-knowledge proof of well-formedness and a signature:

$$[(c_1, \pi_1), \ldots, (c_k, \pi_k)], \pi, \sigma.$$

In this formula we have:

- $c_l = (g^{r_l}, h^{r_l} g^{v_l})$ is the ElGamal encryption of v_l;
- π_l guarantees that v_l is 0 or 1;
- π guarantees that $\sum_{l=1}^{k} v_l = 1$, that is, exactly one voting option has been selected;
- σ is the signature of everything with the voter's key.

Then, to vote, a voter i simply sends her ballot b_i (to the blockchain).

To avoid several kinds of replay attacks, the zero-knowledge proofs and the signatures must be tight to the current election, for instance by including an election identifier in the data that is hashed to derive a challenge.

3.2.3 Tally phase

Computation of the weights. Once the voting phase is over, the first step consists in computing the real weight w'_i of each voter, for the election. This is the voter's base weight (w_i) to which are added all the weights of the delegators, except those who voted:

$$w'_i = w_i + \sum_{\{j \mid j \to i \wedge \nexists b_j \text{ signed by } \mathsf{pk}_j\}} w_j.$$

Computation of the encrypted result. It is assumed that the ballot box has already been cleaned: each ballot $b_i = c_i, \pi_i, \sigma_i$ has a valid zero-knowledge proof and a valid signature, there is at most one ballot per voter (characterized by pk_i) since only the last ballot is kept. The (encrypted) result is then computed by homomorphically combining ballots, weighted by w'_i:

$$c = \prod_i {c_i}^{w'_i}.$$

Decryption. Finally, c is decrypted by the authorities, with a proof of correct decryption, as described in the previous section.

3.3 Variant with coefficiented votes

A voter may not want to vote either 0 or 1 according to her favorite proposal but may instead want to split her vote, e.g. 20% for proposal A, 80% for proposal B, to express a vote that is deemed representative of the opinion of its delegators (or just to reflect appropriately her own personal dilemma).

This can be done quite easily by deciding in advance the granularity p of how a vote can be split. For example, if $p = 100$, a voter could divide her vote in 100 parts, which she can distribute as she sees fit.

The ballot will then be formed with k votes v_l (where k is the number of vote choices), such that $\sum_{l=1}^{k} v_l = p$. Following the notations of Section 3.2.2, the ballot is made up of :

- $c_l = (g^{r_l}, h^{r_l} g^{v_l})$ an ElGamal encryption of v_l;
- π_l guarantees that $v_l \in \{0, \ldots, p\}$;
- π guarantees that $\sum_{l=1}^{k} v_l = p$;
- σ is a signature of all previous data with the voter's key.

The number of votes received by each proposal will therefore be divided by p. Note that the discrete logarithm computation needs to remain reasonable, say $p \sum_{i=1}^{n} w_i < 2^{32}$. Indeed, there exists a discrete logarithm algorithm that requires a number of operations that is essentially the square root of the interval of possible values, and 2^{16} group operations is a matter of seconds at the very worse.

This requires to prove $v_l \in \{0, \ldots, p\}$, which yields a proof of size p times the size of an elementary proof if we use zero-knowledge proof as defined in Belenios. More enhanced techniques exist that yield proofs of size $O(\log(p))$. When p is a power of 2 (for instance, $p = 128$), then the proof is easier.

3.4 A voter can reveal her vote

Let $c = (g^r, h^r g^v)$ be an ElGamal ciphertext of the vote v, where r is a random. Knowing r is enough to decrypt the message, without requiring the decryption key. Therefore, if a voter needs to reveal her vote, she can keep the random values she used in all the ElGamal ciphertexts present in her ballots and publish them. This will provide a proof that she voted as she claims. A simple option would be to use $r = 0$ when the vote must be revealed, so that no further data is needed to reveal the vote, while preserving the algebraic structure of the ballot.

Traditionally, e-voting solutions do not provide tools for doing this, because it is considered as a bad practice to offer to the voters the possibility to prove for whom they voted. In the Tezos context, it might be useful to use this, for instance for the Foundation who wants to prove that they voted "pass". This is a trade-off between secret of the vote and transparency.

3.5 Size and cost estimates

In a typical "yes", "no" or "pass" vote, the ballots described in Section 3.2.2 are composed of three ElGamal ciphertexts and 4 zero-knowledge proofs, 3 of them proving that the ciphertexts contain bits, and an additional proof to show that exactly one bit is set. Each ciphertext takes 2 group elements (3×64 bytes), each bit-ZKP takes 4 $\mathbb{Z}_q$ elements (3×128 bytes), and the additional ZKP takes 2 $\mathbb{Z}_q$ elements (64 bytes). We also need to count the signature, hence 64 more bytes. This sums up to 704 bytes.

It is certainly possible to optimize this by having only two ciphertexts, the first one storing a "yes/no" bit, and a second one storing the "pass" bit. If we impose via a ZKP that when the "pass" bit is set then the first bit is not set, it is still possible to count after a homomorphic operation. The total size of the ZKPs is then 3×128 bytes. This sums up to 576 bytes. We refer to the "Proofs of possibly-blank votes" section of the Belenios specification for an example of a ZKP proof of a disjunction.

The number of exponentiations made by the voter is between one and two dozens, depending on the choices made.

The cost for verifying that a ballot is correct is about the same number of exponentiations.

3.6 Monitoring and context for the blockchain

The security properties of the protocol (in particular with respect to verifiability) relies on the fact that any addition of a ballot can be checked by anyone. We describe here the role of an external auditor, that would have access to the full view of the ballot box. Then the validation of any individual transaction (ballot addition or tally phase) should guarantee that the checks made by an auditor will always succeed.

3.6.1 Monitoring

An auditor has access to:

- the parameters of the election: number of questions, etc;
- the public key of the election, with the (signed) public election key of each trustee (all elements that need to be public according to the DKG protocol);
- the ballot box, that is the list of all ballots accepted so far;
- after the tally: the result and the corresponding proofs of correct decryption.

During the election, it is expected that the auditor:

- verifies the consistency of the ballot box:
  - all ballots are valid (valid zero-knowledge proofs, valid signature for one of the voter keys);
  - no two ballots are signed by the same key.
  - the parameters and the public keys do not change.
- verifies that the ballot box evolves in a way that is in accordance with the protocol: no ballot must disappear.

After the election, the auditor has also access to the result of the election. It is expected that the auditor:

- runs again verifications mentioned above to verify the consistency of the final ballot box;
- verifies that the result is valid w.r.t. the proofs of correct decryption.

3.6.2 Context for the blockchain

The blockchain should guarantee that this monitoring will not detect any issue. Hence, the context of the election must contain:

- the parameters of the election;
- the public keys of the trustees of the election;
- the list of the public keys of voters, with the hash of the corresponding ballot if the voter voted already. This is necessary to allow for revote and vote delegation;
- the current encrypted result c (as defined in Section 3.2.3). Any newly accepted ballot is immediately added (*with the proper weight*) to c to ensure that no ballot disappears.

Remark: Adding a newly accepted ballot *with the proper weight* is actually non trivial:

- The easy case is for a voter i that did not vote so far and did not delegate her Tez to someone else. Then it is sufficient to find the weight w_i associated to i and add her ballot to c with weight w_i.
- To revote, a voter should produce her old ballot (possibly found offchain). This is necessary in order to subtract the contribution of the old ballots from the current encrypted result. The bakers can check that the hash and the signature correspond, and update the hash and c accordingly (subtracting the old ballot and adding the new one).
- In case voter i delegates her Tez to voter j and voter j voted already, she should produce voter j's ballot b_j (possibly found offchain) in addition to her own ballot b_i, so that the bakers can check that the hash and the signature correspond, add the hash of the ballot of voter i, and update c accordingly: subtracting a weight w_i of ballot b_j and adding ballot b_i with weight w_i.
- In case voter i delegates her Tez to voter j and voter j has not yet voted, her ballot is added "normally" to c with weight w_i.
- When a delegate j votes, he should produce the ballots of all delegators i that voted already in addition to his ballot b, so that the bakers can check the hash and the signatures and update c appropriately, that is adding b with a weight

$$w_j \; - \sum_{\substack{i \to j \\ i \text{ voted already}}} w_i.$$

The last three items are required only if the system allows the delegators to override their shares in the vote of their delegate; in a simplified version where this is not possible, these are not relevant.

4 The special case of the proposal phase

4.1 Requirements

The current proposal phase is dynamic. From the beginning to the end of the three weeks, it is possible to propose new amendments, and anyone can endorse as many proposals as they like. This is some kind of approval voting where the proposal that gets the most votes is the winner.

The goal for the next system is to add vote secrecy also for this phase. We do not need to hide the baker who proposes an amendment, but the opinion of the others on the proposals should be secret. If a new proposal is submitted after a voter has cast her ballot, then she can post another one, taking into account this new proposal. The previous ballot is then discarded. At the end of the phase, the trustees should decrypt the result and reveal how many votes were obtained by each proposal.

We accept the following information leakage: if a voter's ballot has been cast before a proposal has been made, then it is clear to everyone that she has not (yet) given her opinion on it.

4.2 A dynamical Belenios

In the classical Belenios setting, the list of questions must be known in advance. This is not what we want for this proposal phase. We propose to adapt it in the following simple way. At any time during the proposal phase a voter can cast a ballot (which will replace any ballot previously cast by the same voter) that contains a list of pairs

$$[id_i, b_i],$$

where id_i is an identifier of a proposal, and b_i is a sub-ballot containing the voter's choice for this proposal. Each sub-ballot must include the encrypted vote and a zero-knowledge proof of well-formedness; furthermore the whole list must be signed.

At the end of the phase, all the sub-ballots with the same proposal identifier are aggregated and decrypted by the trustees to get the number of votes for this proposal.

Some details are still to be decided:

- Do we have a procedure for declaring new proposals, or do we consider that any id that occur in a ballot for the first time is a new proposal?
- Do we require that an incoming ballot includes a sub-ballot for all the known proposals at that time?

The second point is maybe not a good idea. If there are many proposals, it is likely that a voter is only interested in a few of them.

4.3 The threat of Spam

The dynamic scheme that has been sketched is vulnerable to a Spam attack where a malicious baker sends many proposals. Then the size of the ballots will increase accordingly if the tool used by the voter always includes a sub-ballot for each known proposal.

We suggest the following mitigation: any ballot can include at most 10 sub-ballots. If there are more than 10 proposals, then we assume that we are subject to a Spam situation, and there are still at most 10 proposals that the voter will consider to be serious. Then the voter will only include sub-ballots for these non-Spam proposals.

Due to the quorum rule, the Spam proposals won't have any chance to win the proposal phase, and the effect on the size of the ballots is limited to a factor of 10.

In this paragraph, the number 10 is of course an arbitrary number that can be fixed to any value, taking into account the ecosystem.

Privacy risk. This procedure comes with a risk in terms of privacy in the following situation. We assume that there are two serious proposals and hundreds of Spam proposals. We also assume that the voter would say "yes" to the first one and "no" to the second one. The interface of the voting client will have to show all the proposals to the voter and let her decide the proposals that she sees as serious and let her give her opinion on them. If the interface is not clear enough, we fear a risk that the voter will just select the proposal that she prefers, and does not make the effort to select the ones for which she wants to vote "no". She might think that this is even better if she does not contribute to the quorum for the second serious proposal. However, in this case, she leaks that she didn't vote "yes" for this second proposal.

Therefore, it is important to have a user interface that is very clear about the privacy risk of not giving an opinion on a non-spam proposal.

References

[1] Belenios – Verifiable online voting system. `https://gitlab.inria.fr/belenios/belenios`, 2020. Version 1.13.

[2] V. Cortier, D. Galindo, S. Glondu, and M. Izabachene. Distributed ElGamal à la Pedersen - application to Helios. In *Workshop on Privacy in the Electronic Society (WPES 2013)*, Berlin, Germany, 2013.

[3] O. de Marneffe, O. Pereira, and J.-J. Quisquater. Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios. In *2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections (EVT/WOTE 2009)*, Montreal, Canada, 2009.

[4] S. Glondu. Belenios specification - version 1.11. `http://www.belenios.org/specification.pdf`, 2018.

[5] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In *Workshop on Privacy in the Electronic Society (WPES 2005)*, page 61–70. ACM, 2005.