Our protocol is a combination of @schoenmakersLectureNotesCryptographic2018 and @haoAnonymousVotingTworound2010.
For the threshold 3-round voting scheme we use the technique described in @schoenmakersLectureNotesCryptographic2018 (link, section 7.1 Electronic Voting, page 78); however, we use elliptic-curves, zkSNARK proofs, and multi-candidates settings.
For multi-candidates, we use the technique described in @haoAnonymousVotingTworound2010 (link).

Assumptions:

- All communication is done over a public message board (blockchain).
- Authenticated public channels are available for every participant, achieved by a public message board and digital signatures.
- The set of all n participants $\vec{P} = P_i, \ldots, P_n$ is publicly known.
- Each party P_i consists of key pair (s_i, P_i), where $s_i \in_R \mathbb{Z}_q$ is a randomly selected secret key, and $P_i = s_i \times G$ is the corresponding public-key. We use the same notation for party and public key, P_i, as parties are identified by their public keys only.
- We use Elliptic Curve Cryptography, specifically babyJubJub curve.
- Participation in the protocol is equivalent to agreeing to:
    - Elliptic Cuve $E(\mathbb{Z}_q)$, with the base (generator) point on curve G;

    - set of eligable voters (participants) $\vec{P} = P_i, \ldots, P_n$, where n is the number of all voters;
    - candidate options $C_1, \ldots, C_l$, where l is the number of all candidates.
- Public-key encryption is realised using ElGamal cryptosystem. $EG_P(\cdot)$ is the ecnryption algorithm for public key P, and $EG_s(\cdot)$ is the decryption algorithm using corresponding secret key s.

Round 1. Distributed Key Generation

Secret Sharing

The goal of DKG protocol is to jointly generate key-pair without any party learning the secret key. Each party $P_1, \ldots, P_n$ learns only its share of the secret key, while public key is publicly known.
Sharing a secret can be done using Shamir Secret Sharing (SSS), which allows a dealer to encode secret key s into a random polynomial $f(X) = a_0 + a_1 X + a_2 X^2 + \cdots + a_{t-1} X^{t-1}$, where $a_0, a_1, \ldots, a_{t-1} \in_R \mathbb{F}_q$; the secret key $s = a_0$ and $t - 1$ is the degree of polynomial. Following a Lagrage Theorem, t number of points on the polynomial $f(X)$ allows for reconstructing the polynomial and hence extract secret key by computing $s = f(0)$. The shares are distributed to parties $P_i, 1 \leq i \leq n$, by evaluating function at a corresponding point $f(i)$. The polynomial of degree t can be reconstructed with $t + 1$ points using Lagrange Interpolation.

Distributed Key Generation

Since we don't want any party to become a dealer (and learn the secret key), we have to distribute the generation of polynomial $\mathbf{f}(X) \in_R \mathbb{Z}_q[X]$ across parties. It is done by having each party pick a random polynomial $f_i(X) \in \mathbb{Z}_q[X]$, and then define the final polynomial $\mathbf{f}(X) = \sum_{i=1}^{n} f_i(X)$; hence the voting decryption (secret) key $\mathbf{d} = \mathbf{f}(0)$, and voting encryption (public) key $\mathbf{E} = \mathbf{d} \times G$. Additionally, to prevent misbehavior of parties (sending arbitrary values) we use more sophisticated version of SS called Publicly Verifable Secret Sharing (PVSS) which involves zero-knowledge proofs attesting that the correct relation between values holds.

Every party can (but does not have to) participate in the DKG phase. The actual number of parties that participate is denoted by m where the maximum number is n. Since we focus on small scale votings where participants know each other, we make a social assumption, that each participant trusts k other parties.

The DKG protocol involves each party $P_i \ldots, P_m$, $i \leq m \leq n$:

- Chose a set of k trusted parties.
- Sample random polynomial $f_i(X) \in_R \mathbb{Z}_q[X]$ of degree $t-1$, where $t \approx 0.6k$.
- Compute decryption (secret) key $d_i = f_i(0)$ and encryption (public) key $E_i = d_i \times G$.
- Compute zero-knowledge proof (ZKP) of exponent $D_i = d_i \times G$ using Schnorr's signature. Namely, the proof is $\sigma_i = (r \times G, k = r - d_i \times c)$, where $r \in_R \mathbb{Z}_q$ and $c = H(G, i, r \times G, d_i \times G)$.
- Compute shares of decryption key $\vec{d_i} := \{f_i(j) : j \in \{1 \ldots k\}\}$, and encrypt each share to each

corresponding trusted party
$\vec{EG}_{\vec{P}}(\vec{d}_i) = \{EG_{P_j}(\vec{d}_i[j]) : j \in \{1 \dots n\}\}$.

- Compute zero-knowledge proof of elGamal encryption showing that $EG()$ encrypts a share $\vec{d}_i[j]$ as described in verifable secret sharing (PVSS). Namely, TODO.
- Broadcast tuple of public key, zkp, and all encrypted shares $(E_i, \sigma_i, \vec{EG}_{\vec{P}}(\vec{d}_i))$ to message board.

State after Round 1.

After the DKG has completed (once it reached n messages or after some predefined period). The message board state looks as follows:

- $\{E_i : 1 \leq i \leq m\}$, shares of voting public key.
- $\{\sigma_i : 1 \leq i \leq m\}$, proofs of exponents.
- $\{\{EG_{P_j}(\vec{d}_i[j]) : j \in \{1 \dots n\}\} : 1 \leq i \leq m\}$, encrypted shares of shares of voting secret key.

The voting encryption key $\mathbf{E}$ can be reconstructed by everyone by computing $\mathbf{E} = \sum_{i=1}^{n} E_i$.

The share of voting decryption key $\mathbf{D_i}$ can be reconstructed by party P_i by computing $\mathbf{D}_i = \sum_{j=1}^{m} EG_{s_i}(d_{ji})$.

Round 2. Voting

Every party can (but does not have to) participate in the voting phase. The actual number of parties that participated is denoted by k where the maximum number is n.

One-candidate

The voting phase involves each party $P_i \ldots, P_k$:

- Select a vote $v_i \in \{0, 1\} \simeq \{\text{"no"}, \text{"yes"}\}$.
- Create an encrypted ballot B_i using ElGamal encryption. $B_i = (k_i \times G, k_i \cdot \mathbf{E} + v_i \times H)$, where $k_i \in_R \mathbb{Z}_q$ is a blinding factor for user i, and G and H are public parameters.
- Compute a Σ-proof σ_i that B_i is correctly formed, namely, $v_i \in \{0, 1\}$. TODO.
- Broadcast (B_i, σ_i).

Multiple candidates

Let $C_1 \ldots C_l$ be a set of all possible candidates.

The voting phase involves each party $P_i \ldots, P_k$:

- Select a vote $v_{ij} \in \{0, 1\} \simeq \{\text{"no"}, \text{"yes"}\}$ for each candidate $j \in \{1 \ldots l\}$.
- Compute a ballot using ElGamal encryption for $B_i = (r_i \times G,\ r_i \cdot \mathbf{E} + v_{i1}C_1 + \cdots + v_{il}C_l)$, where
  - $r_i \in_R \mathbb{Z}_q$ is a blinding factor for user i, and
  - $C_{1,} \ldots, C_l$ are independent generators (one for each candidate).
- Compute a Σ-proof that B_i is correctly formed. Namely that each $v_{ij} \in \{0, 1\}$ and that $\sum_{j=1}^{l} v_{ij} = 1$. TODO.
- Broadcast (B_i, σ_i).

State after Round 2.

After the voting phase has completed (once it reached n messages or after a predefined period). The message board state is appended by:

- $\{(B_i, \sigma_i) : 1 \leq i \leq k\}$, set of encrypted votes casted by k voters, along with ZKPs showing that v_{ij} is one of $\{0, 1\}$.

Round 3. Online Tallying

The decryption and tally is achieved via (t, m) Threshold ElGamal Decryption. The tallying phase involves any trusted subset $t = 0.6k_i$ of each participant from phase 1 $P_i \ldots, P_t$:

- Sum the first part of the ballots (aka. shared keys) $A = \sum_{i=1}^{k} r_i \times G$.
- Multiply it with the share of the decryption key $A_i = \mathbf{d}_i \times A = \mathbf{d}_i \times G \times \sum_{i=1}^{k} r_i$.
- Broadcast A_i to message board.

State after Round 3.

After the voting phase has completed—once it reached $t \leq m$ messages. The message board state is appended with:

- $\{A_i : 1 \leq i \leq t\}$, set of blinded shares of decryption keys.

Offline Tallying

Single-candidate tally

Everyone can calculate:

- Compute $Z = \sum_{i=1}^{k} A_i \times \Pi_{j=1}^{t} \frac{j}{j-i}, i \neq j$.
- Sum of the second part $B = \sum_{i=1}^{k} (r_i \times \mathbf{E} + v_i \times C)$.
- The decryption of the partial result is $M = B - Z = C \times \sum_{i=1}^{k} v_i$, because:

$$
\begin{aligned}
M &= B - Z \\
&= \sum_{i=1}^{k}(r_i \times \mathbf{E} + v_i \times C) - Z \\
&= \sum_{i=1}^{k}(r_i \times \mathbf{E} + v_i \times C) - \sum_{i=1}^{k} \mathbf{A_i} \times \mathbf{\Pi}_{j=1}^{t} \frac{j}{j-i} \\
&= \sum_{i=1}^{k}(r_i \times \mathbf{E} + v_i \times C) - \sum_{i=1}^{k} \mathbf{d}_i \times G \times \sum_{i=1}^{k} r_i \times \mathbf{\Pi}_{j=1}^{t} \frac{j}{j-i} \\
&= \sum_{i=1}^{k}(r_i \times \mathbf{E} + v_i \times C) - G \times \sum_{i=1}^{k} r_i \times \sum_{i=1}^{k} \mathbf{d}_i \times \mathbf{\Pi}_{j=1}^{t} \frac{j}{j-i} \\
&= \sum_{i=1}^{k}(r_i \times \mathbf{E} + v_i \times C) - G \times \sum_{i=1}^{k} r_i \times \mathbf{d} \\
&= \sum_{i=1}^{k}(r_i \times \mathbf{E} + v_i \times C) - \sum_{i=1}^{k} r_i \times \mathbf{E} \\
&= \sum_{i=1}^{k}(v_i \times C) + \sum_{i=1}^{k}(r_i \times \mathbf{E}) - \sum_{i=1}^{k} r_i \times \mathbf{E} \\
&= \sum_{i=1}^{k}(v_i \times C) \\
&= C \times \sum_{i=1}^{k} v_i
\end{aligned}
$$

- The total number of "yes" votes is $x = \sum_{i=1}^{k} v_i$.
- To extract x from $M = x \times C$ we have to solve Elliptic-Curve Discrete Logarithm Problem. Fortunatelly, since the value of x is small, i.e., in range $[0, k]$, we can use exhaustive search or Shanks' baby-step giant-step algorithm.

Multi-candidate tally

Everyone can calculate:

- Compute $Z = \sum_{i=1}^{k} A_i \times \Pi_{j=1}^{t} \frac{j}{j-i}, i \neq j$.
- Sum of the second part
  $B = \sum_{i=1}^{k} (r_i \times \mathbf{E} + v_{i1} C_1 + \cdots + v_{il} C_l)$.
- The decryption of the partial result is
  $M = B - Z = C_1 \times \sum_{i=1}^{k} v_{i1} + \cdots + C_l \times \sum_{i=1}^{k} v_{il}$, because:

$$
\begin{aligned}
M &= B - Z \\
&= \sum_{i=1}^{k} (r_i \times \mathbf{E}) + C_1 \times \sum_{i=1}^{k} v_{i1} + \cdots + C_l \times \sum_{i=1}^{k} v_{il} - Z \\
&= \sum_{i=1}^{k} (r_i \times \mathbf{E}) + C_1 \times \sum_{i=1}^{k} v_{i1} + \cdots + C_l \times \sum_{i=1}^{k} v_{il} - \sum_{i=1}^{k} r_i \times \mathbf{E} \\
&= C_1 \times \sum_{i=1}^{k} v_{i1} + \cdots + C_l \times \sum_{i=1}^{k} v_{il}
\end{aligned}
$$

- The total number of "yes" votes for cadidate c is
  $x_c = \sum_{i=1}^{k} v_{ci}$.
- To extract x_c from $M = x_1 \times C_1 + \cdots + x_l \times C_l$ we have to solve Elliptic-Curve Discrete Logarithm Problem. To extract each x_i we use the technique described in @haoAnonymousVotingTworound2010.