# Anonymous Online Voting with Multiple Candidates

Michael Youssef
*CSCI 1515*

(Dated: May 16, 2023)

Anonymous online voting requires that voters be able to anonymously cast a vote (0 or 1) without revealing their vote to any party, including the system's arbiters and tallyers. This can be achieved using Threshold Encryption via. an additively homomorphic encryption scheme like ElGamal. This protocol also uses non-interactive zero-knowledge proofs (NIZK) to prove the validity of votes by voters and partial decryptions by arbiters. This protocol is extended to allow candidates to vote for at most $k$ candidates by introducing another NIZK that must be sent from voters.

## I. PREREQUISITES

An anonymous online voting system consists of a registrar, a tallyer, multiple non-colluding arbiters, and the voters.

**Registrar**:

The registrar registers voters and issues each voter a certificate. This certificate is signed over a voter's id and his/her public verification key, which is used by the tallyers and arbiters to confirm the authenticity of any messages sent from that voter.

**Voters**:

Each voter must submit a 0-or-1 vote and prove this without revealing their actual vote. This is done using a ZKP where a voter's ciphertext is $(c_1, c_2) = (g^r, g^v pk^r)$ where $r$ is some randomness and $v \in \{0, 1\}$. With $r$ as the witness, each voter provides a Disjunctive Chaum-Pederson (DCP) ZKP.

**Tallyer**:

The tallyer is responsible for checking the ZKP provided by a voter (including make sure that he/she has not voted more than once) and publishing their vote/ZKP to a database that is shared with the arbiters.

**Arbiters**:

The arbiters are mainly responsible for generating a combined public key and jointly decrypting votes via. threshold ElGamal encryption.

More specifically, each arbiter generates a keypair $(sk_i, pk_i)$ where $pk_i = g^{sk_i}$; thus, the combined public key becomes $pk = \prod_{i=1}^{n} pk_i = g^{\sum_{i=1}^{n} sk_i}$. To partially decrypt a combined ciphertext $(c_1, c_2)$, the $ith$ arbiter computes $d_i = c_1^{sk_i}$.

To ensure that each arbiter provides a valid partial decryption $d_i$, they must also submit ZKPs that prove knowledge of the witness $sk_i$ in the Diffie-Hellman tuple $(g^{sk_i}, c_1, c_1^{sk_i})$. Upon verifying each arbiter's ZKP, the voters can then use $d = \prod_{i=1}^{n} d_i = c_1^{sk}$ to decrypt $c_2$ and determine the number of votes for a single candidate.

## II. OVERVIEW

The voter registration in a single-candidate voting system remains in place, but several modifications need to be made to the messages sent between voters and tallyers, tallyers and arbiters, and arbiters to voters.

**Modifications**:

In a ballot with $k$ candidates, each voter must provide a vote-ZKP pair for each candidate. Thus, a voter's ballot should be a k-tuple of tuples: $((g^{r_1}, pk^{r_1} g^{v_1}), \ldots, (g^{r_k}, pk^{r_k} g^{v_k}))$. It is crucial that the voter does not use the same $r$ for encrypting each vote because this will uniquely distinguish votes where $v = 0$ from those where $v = 1$.

To ensure that a voter votes for no more than $k$ candidates, a voter must also provide a Disjunctive Chaum-Pederson ZKP proving that he/she voted for $0, 1, \ldots,$ or $k$ candidates. The voter stores $r = \sum_{i=1}^{k} r_i$ and uses it as a witness to prove that $v = \sum_{i=1}^{k} (v_i) \leq k$ in the tuple $(a, b) = (g^r, pk^r g^v)$.

**Disjunctive Chaum-Pederson**:

Suppose a voter for $\tilde{k} \leq k$ candidates. For each $v \in [0, k]$, if $v \neq \tilde{k}$, the voter computes the following:

1. Randomly sample $c_v$ and $r''_v$ from $Z_q$
2. $b' = b/g^v \pmod{\text{p}}$
3. $a'_v = g^{r''_v}/a^{c_v} \pmod{\text{p}}$
4. $b'_v = pk^{r''_v}/(b')^{c_v} \pmod{\text{p}}$

For $v = \tilde{k}$, the following are computed:

1. Randomly sample $r'_{\tilde{k}}$ from $Z_q$
2. $a_{\tilde{k}} = g^{r'_{\tilde{k}}} \pmod{\text{p}}$
3. $pk_{\tilde{k}} = pk^{r'_{\tilde{k}}} \pmod{\text{p}}$
4. $c_{\tilde{k}} = H(pk, a, b, \{a'_i\}_{i=1}^{k}, \{b'_i\}_{i=1}^{k}) - \sum_{i \neq \tilde{k}} c_i$.
5. $r''_{\tilde{k}} = r'_{\tilde{k}} + (c_{\tilde{k}})(r) \pmod{\text{q}}$

For each $v \in 0, 1, \ldots, k$, the tallyer verifies the following:

1. $g^{r''_v} = (a'_v)(a^{c_v}) \pmod{p}$
2. $pk^{r''_v} = (b'_v)(b/g^v)^{c_v} \pmod{p}$

Finally, the tallyer checks if $\sum_{i=1}^{k} c_i = H(pk, a, b, \{a'_i\}_{i=1}^{k}, \{b'_i\}_{i=1}^{k})$.

## III. DIFFICULTIES

The entire protocol was implemented in C++ using the Crypto++ library. Extending the baseline protocol required modifying the message structs used in communication, the SQL databases used to hold voter's votes as well as partial decryptions from the arbiters.

The voter-tallyer and tallyer-arbiter needed to store vectors of votes/ZKPs and needed a new field for the vote count ZKP. Additionally, the partial decryptions and their accompanying ZKPs needed to be stored in vectors. This required changing most of the existing message structs and their serialization / deserialization functions. Although a lot of the logic of the baseline protocol was left intact, this nevertheless required rewriting most of the existing methods. These excessive modifications tended to be a major source of obscure bugs.

## IV. RESOURCES

Note: I also consulted the existing source code in the project that was provided in the CSCI 1515 TA staff as well as lectures 11-13.

**References**:

Bernard et al., *Cryptographic Voting - A Gentle Introduction*, University of Bristol, England.