Overview of bulletin boards for e-voting

Annabell Kuldmaa
Supervised by Helger Lipmaa

December 16, 2016

1 Introduction

In this report we consider the task of implementing secure e-voting systems. This is an active area of research due to the fact that e-voting is expected to reduce the cost of elections and increase the voter turnout. Furthermore, the goal is to verify the entire election procedure.

In the high-level description we may divide the server-side of voting into three main stages. In the first stage the task is to receive the ballots and store them until the end of the election period. In the second stage we need to anonymize the ballots and in the last stage we must decrypt the encrypted ballots. These steps strongly depend on the approach chosen, e.g., in case of code-voting the first and the second steps cannot be distinguished.

The shuffling stage can be implemented using mix-nets. Mix-net is a set of set of mix servers that guarantee the anonymity by mixing the input in a verifiable way. Many implementations of mix-nets suitable for this scenario have been introduced, e.g., [BG12] and it is stated that to guarantee the anonymity of voters at least one of the mix servers must be uncorrupted.

The first step can be implemented using bulletin boards, i.e., we require that a corrupted server cannot modify the ballots and the storage of the votes must be done in a verifiable manner. The authors of [CGS97] claim that bulletin board can be viewed as a secure broadcast channel and, therefore, the bulletin board can be implemented as a set of replicated servers implementing a Byzantine Agreement protocol. Although the latter claim is widely spread, the implementation of a bulletin board is not straightforward. In fact, there are only a few approaches introduced which in fact are not purely Byzantine Agreement protocols.

In this paper we review two approaches of how a bulletin board can be implemented in the context of e-voting. We begin in Section 2 with formally defining the Byzantine Agreement problem and properties of bulletin board. In Section 3 we present a bulletin board implementation introduced by Culcane and Schneider in [CS14]. This is followed in Section 4 with describing a voting system introduced in [CZZ+15] that also includes a bulletin board implementation.

2 Preliminaries

In this Section we introduce the Byzantine Agreement and Byzantine Generals problem. Also, we formally define the requirements for a bulletin board.

We begin with defining the Byzantine Generals problem.

Definition 2.1 (Byzantine Generals [LLR06]) *Let $P_1, \ldots, P_{n-1}, P_n$ be n parties. Assume $P_n = G$ is a designated party with input $x \in \{0, 1\}$ and let $\mathcal{A}$ be an adversary controlling up to $n - t$ parties. We say that a protocol solves the Byzantine Generals problem tolerating $n - t$ corrupted parties, if for any adversary $\mathcal{A}$ the following holds:*

- *all honest parties P_i must output the same value;*
- *if G is honest, then all honest parties P_i must output x;*
- *all honest parties must output some value.*

Next, let us formally define the Byzantine Agreement problem. Note that the main difference is that if honest parties begin with the same input, the output must be that particular value.

Definition 2.2 (Byzantine Agreement [LLR06]) *Let $P_1, \ldots, P_n$ be n parties with inputs $x_1, \ldots, x_n$ respectively. Let $\mathcal{A}$ be an adversary controlling up to $n - t$ parties. We say that a protocol solves the Byzantine Agreement problem tolerating $n - t$ corrupted parties, if for any adversary $\mathcal{A}$ the following holds:*

- *all honest parties P_i must output the same value;*
- *if more than $\frac{n}{2}$ parties are honest and have same input value, they must all output that particular value;*
- *all honest parties must output some value.*

We have that any protocol that solves the Byzantine Generals problem also solves the Byzantine Agreement problem. The other direction holds, if assume that the majority of the parties are honest.

Now, let us describe the properties we require from a bulletin board presented in [CS14]. We want that the following is satisfied:

- only items that have been posted can appear on the bulletin board;
- all valid votes are published;
- two clashing items cannot be published;
- after publishing, any item cannot be removed.

3 A Bulletin board by Culnane and Schneider

In this Section we describe a bulletin board implementation introduced by Culcane and Schneider in [CS14]. They propose a rather simple bulletin board implementation, motivated by the vVote verifiable voting system [CRST14].

Denote by n the total number of participating parties and by t the total number of honest parties. Assume that $t > \frac{2}{3}n$. In the following we use party and peer interleaved.

The setting is as follows: each voter receives a receipt if vote is accepted and the public bulletin board is updated in period p, e.g., every hour. The protocol uses (n, t)-threshold signature scheme, i.e., at least t shares are needed to create a valid signature.

Denote peer i's threshold signing key by ssk_i and individual signing key by sk_i. Furthermore, denote by $sig_k(m)$ a signature on m using key k.

The protocol (Figure 1) for posting votes is as follows:

1. voter sends vote v to each peer $i = 1, \ldots, n$;
2. each peer i checks that there are no clashes with previous posts and sends $sig_{sk_i}(v)$ to all other servers;
3. upon receiving t signatures, server i sends $sig_{ssk_i}(v)$ to the voter;

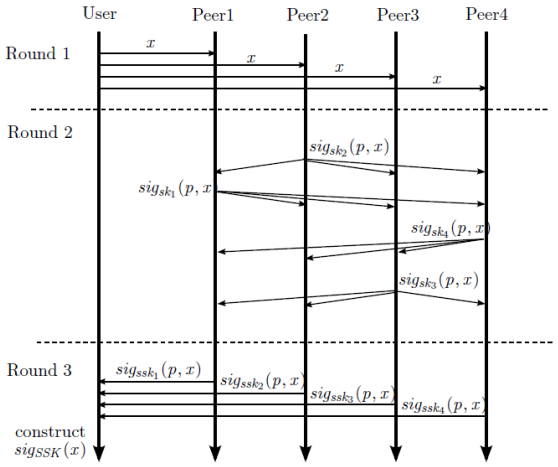

Figure 1: Posting protocol

4. upon receiving t threshold signatures, the voter combines them into a receipt.

Denote by $B_{i,p}$ the table of peer i's votes received in period p. Note that we store in the latter table only votes that the peer i also received a t signatures from other peers and sent the threshold signature to the voter. Moreover, denote by $D_{i,p}$ the corresponding signatures received from other peers.

The protocol for publishing the bulletin board is as follows:

1. each peer i sends $sig_{sk_i}(B_{i,p})$ to peer j;
2. if t peers agree, each peer i publishes $sig_{ssk_i}(B_{i,p})$ and $B_{i,p}$ to the bulletin board.

The protocol is also presented as Figure 2.

Next, consider a scenario where t servers do not agree. If that is the case, each server i sends $D_{i,p}$ to every other server j and every server j updates its table. The intuition behind this step is that every server j will update its database with missing signatures and as $D_{i,p}$ consists of signed honest parties will not update databases with fake items. After that parties will repeat the protocol for publishing votes to the bulletin board. Notice that all honest parties will agree on the database after the maximum of $n - t + 1$ rounds of fall-back protocol.

The authors of [CS14] use Event-B framework to prove the correctness of the protocol with respect to the properties of bulletin board. The idea is to describe the system as in terms of states and events that transform the state. We refer the interested reader to [CS14] for the detailed approach.

The authors of [CS14] claim that their implementation is more efficient than using a Byzantine Agreement protocol. What is more, they state that Byzantine agreement would be too inefficient for receiving large quantity of votes and require more rounds. On the other hand, their protocol for publishing stage is similar to a Byzantine Agreement protocol introduced in [Lyn96].

4 Bulletin boards and D-DEMOS

In this Section we give an overview of D-DEMOS which is a complete e-voting system that is distributed, privacy-preserving and end-to-end verifiable. It was introduced by Chondros et al in [CZZ+15]. D-DEMOS uses vote codes, i.e., mixing is done in the initialization phase.

Their systems consists of the following sub-systems:

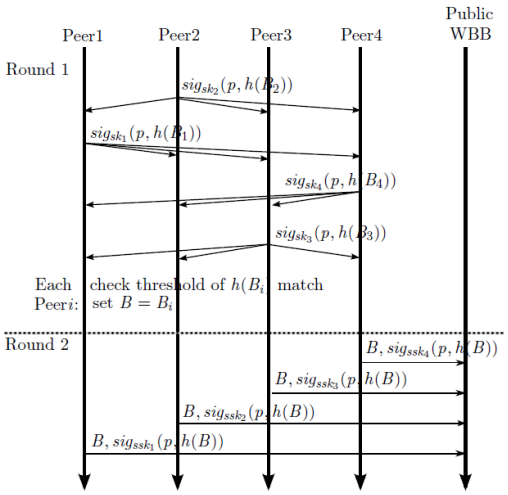

Figure 2: Publishing protocol

- Election Authority (EA): it initializes all remaining system components in the set-up phase;
- Vote-Collection (VC) sub-system: it collects votes from voters during the election;
- Bulletin Board (BB) sub-system: it stores ballots, votes and the result, allows any party to read from it and verify the election process;
- Trustees: hold keys to uncover information from BB.

Note that during voting period ballots are stored in VC nodes and votes are published to BB nodes after the election end. Thus, in the following we introduce the protocols for both VC and BB nodes.

Each voter has a unique ballot consisting of a unique number denoted by ”serial-no” and of m vote codes for each candidate.

4.1 Vote Collector sub-system

VC is a subsystem consisting of N_v nodes. In order to tolerate Byzantine failures the number of corrupted nodes must be strictly less that $\frac{1}{3}$ of the total VC nodes. The number of corrupted VC nodes is denoted by f_v.

These nodes have private and authenticated channels between each other and a public channel to the voters. VC nodes run two protocols: voting and vote-set consensus.

The voting protocol is presented as Algorithm 1.

Algorithm 1 Vote Collector algorithms

```
1:  procedure ON VOTE(serial-no, vote-code) from source:
2:      if SysTime() between start and end
3:          b :=locateBallot(serial-no)
4:          if b.status == NotVoted
5:              l := ballot.VerifyVoteCode(vote-code)
6:              if l ≠ null
7:                  b.UCERT := {}                    ▷ Uniqueness certificate
8:                  sendAll(ENDORSE⟨serial-no, vote-code⟩)
9:                  wait for (N_v − f_v) valid replies, fill b.UCERT
10:                 b.status := Pending
11:                 b.used-vc := vote-code
12:                 b.lrs := {}                      ▷ list of receipt shares
13:                 sendAll(VOTE_P⟨serial-no, vote-code, l.share⟩)
14:                 wait for (N_v − f_v) VOTE_P messages, fill b.lrs
15:                 b.receipt := Rec(b.lrs)
16:                 b.status := Voted
17:                 send(source, b.receipt)
18:         else if b.status == Voted AND b.used-vc == vote-code
19:             send (source, ballot.receipt)

20: procedure ON VOTE_P(serial-no, vote-code, share, UCERT) from
    source:
21:     if UCERT is not valid
22:         return
23:     if SysTime() between start and end
24:         b :=locateBallot(serial-no)
25:         if b.status == NotVoted
26:             l := ballot.VerifyVoteCode(vote-code)
27:             if l ≠ null
28:                 b.status := Pending
29:                 b.used-vc := vote-code
30:                 b.lrs.Append(share)
31:                 sendAll(VOTE_P(serial-no, vote-code, l.share) )
32:         else if b.status == Voted AND b.used-vc == vote-code
33:             b.lrs.Append(share)
34:             if size(b.lrs) >= N_v − f_v
35:                 b.receipt := Rec(b.lrs)
36:                 b.status := Voted

37: function BALLOT::VERIFYVOTECODE(vote-code)
38:     for l = 1 to ballot_lines do
39:         if lines[l].hash == h(vote-code||lines[l].salt) return l
        return null
```

The overall idea is as follows:

1. a voter sends a VOTE(serial-no, vote-code) to a random VC node, denoted in the following by responder;
2. the responder checks if the request was within the election time;
3. the responder locates the ballot given ”serial-no”;
4. the responder checks this ballot has not been used;
5. the responder compares the ”vote-code” against every hashed vote code in each ballot line until the correct entry is located;
6. the responder sends to all other VC nodes ENDORSE(serial-no, vote-code);
   - each VC node upon receiving ENDORSE (serial-no, vote-code) checks that it has not endorsed another ”vote-code” for this ballot, then signs ”serial-no” and ”vote-code” denoted by sig_{VC_i}, and sends ENDORSEMENT(serial-no, vote-code, sig_{VC_i}) to the responder;
7. the responder waits for $N_v - f_v$ valid signatures and forms UCERT;

8. the responder obtains "receipt-share" from its local database and sends VOTE_P(serial-no, vote-code, receipt-share, UCERT) to all other VC nodes;
    - upon receiving a VOTE_P message, VC node validates the UCERT and "receipt-share";
    - each node does the same checks as the responder;
    - each node sends VOTE_P(serial-no, vote-code, receipt-share) to the responder where "receipt-share" is obtained from its local database;
9. the responder waits for $N_v - f_v$ valid shares and reconstructs a receipt for the voter;
10. the responder marks the ballot as voted and sends the receipt to the voter.

After the end of the election, VC nodes follow the vote-set consensus protocol consisting of the following steps for each registered ballot:

1. send ANNOUNCE(serial-no, vote-code, UCERT) to all other VC nodes;
2. wait for $N_v - f_v$ such messages and if any of these contains a valid "vote-code" and UCERT, change the local state for this ballot;
3. perform Binary Consensus protocol (1 if there was a valid vote for this ballot, otherwise 0);
4. if the output of the protocol above is 0, there was no vote for this ballot;
5. if the output is 1, there are two cases:
    (a) if the vote code has a valid UCERT, the ballot is considered to have voted for the specified vote-code;
    (b) if the vote code is not known, a VC node sends a recover request, waits for a first valid response and updates its local database.

The output of the Binary Consensus protocol is translated to not-voted/has a unique valid vote code.

The authors of [CZZ+15] consider the following case. Assume that a voter submitted a valid vote code, but a receipt was not generated before the election end time.

Therefore, an honest VC node might not be aware of the vote code and will enter the agreement protocol with input 0. By the definition of Byzantine Agreement 2.2, the result of the consensus algorithm can be random, either 0 or 1. If the result is 1, then there exists a VC note that has a valid certificate and a vote code. The respond protocol (5b) guarantees, that an honest node will get the certificate and the vote code. If the result is 0, vote is not counted in the final tally. As a result, there are no guarantees for a vote without a receipt whether it is in the final tally or not.

We propose that issue can be overcome by sending the ENDORSEMENT in Step 6 (Algorithm 1) not only to the responder but all other VC nodes. That way every VC node receives a UCERT already at this particular step. Furthermore, by doing that we do not need to perform the Binary Consensus protocol. Currently this is still an assumption and needs to be proven.

4.2 Bulletin Board

BB is a subsystem consisting of N_b nodes. The number of corrupted BB nodes is denoted by f_b and this must be strictly less than $\frac{1}{2}$.

Each bulletin board initially stores msk generated by Election Authority, i.e., for each voter stores hashes of encrypted vote codes.

Writing to a BB node can only be done through an authenticated channel, but reading can be done via a public and anonymous one. Note that BB nodes do not directly communicate with each other, i.e., readers issue the read request to all BB nodes trusting the reply that comes from the majority and writers write to all BB nodes.

An overview of the functionality of a BB node is described as follows:

1. after set-up, each BB node publishes its initialization data;
2. during election all BB nodes remain inert;
3. after voting, each BB node receives the final vote-code set and the shares of msk from each VC node;
4. once receiving $f_v + 1$ identical vote-code sets, a BB node publishes the vote-code set;
5. once receiving $N_v - f_v$ valid key shares, a BB node reconstructs msk, decrypts vote-codes and publishes them;
6. after trustees have validated data, BB nodes calculate and publish the final election result.

The procedure of validating the result is further described in [CZZ+15].
The authors of [CZZ+15] give a security proof for the following properties:

- liveness;
- safety (every honest voter who receives a valid receipt is assured that her vote will be published on the honest BB nodes and included in the election tally with probability at least $1-\mathrm{negl}(\lambda)-\frac{f_v}{2^{64}-f_v}$;
- end-to-end verifiability;
- voter privacy (assume that adversary controls the entire VC subsystem, f_b BB nodes, $f_t < \frac{N_t}{3}$ trustees),

where λ denotes the security parameter.

5 Conclusions and Future Work

In this report we studied two different approaches for implementing bulletin boards in the context of e-voting: a bulletin board implementation by Culcane and Schneider and D-DEMOS. The latter system is much more complicated as it is designed very specifically for a concrete voting system. Furthermore, in D-DEMOS the functionality of a bulletin board is divided into sub-systems (VC nodes and BB nodes), and voter communicates with only one VC node. Also, in D-DEMOS the ballots are posted to multiple BB nodes, but in the approach of Culcane and Schneider there is one public bulletin board.

On the other hand, there are also similarities. For example, the receipt is generated using threshold signatures and we assume that at least $\frac{2}{3}n$ of peers are honest.

In general, the open question is whether it is possible to get a better bound for the number of honest parties. In fact, the assumption that at least $\frac{2}{3}$ of parties must be honest is relatively strict. For example, if we consider mix-servers, if at least one mix-server is honest, then the anonymity is guaranteed. We are also interested in a bulletin board implementation, that would be suitable for client-side encryption voting systems with provable security.

What is more, interesting question is whether it is possible to combine the functionality of a bulletin board and mix-nets. Until now we have not noticed any work in this area although it is very relevant.

References

[BG12] Stephanie Bayer and Jens Groth. *Efficient Zero-Knowledge Argument for Correctness of a Shuffle*, pages 263–280. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. *A Secure and Optimally Efficient Multi-Authority Election Scheme*, pages 103–118. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[CRST14] Chris Culnane, Peter Y. A. Ryan, Steve Schneider, and Vanessa Teague. vvote: a verifiable voting system (DRAFT). *CoRR*, abs/1404.6822, 2014.

[CS14] Chris Culnane and Steve Schneider. A peered bulletin board for robust use in verifiable voting systems. *CoRR*, abs/1401.4151, 2014.

[CZZ+15] Nikos Chondros, Bingsheng Zhang, Thomas Zacharias, Panos Diamantopoulos, Stathis Maneas, Christos Patsonakis, Alex Delis, Aggelos Kiayias, and Mema Roussopoulos. A distributed, end-to-end verifiable, internet voting system. *CoRR*, abs/1507.06812, 2015.

[LLR06] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. *J. ACM*, 53(6):881–917, November 2006.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.