U.C. Berkeley — CS276: Cryptography | Lecture 26
Professor: David Wagner | April 20, 2006
Scribe: Ilya Bagrak

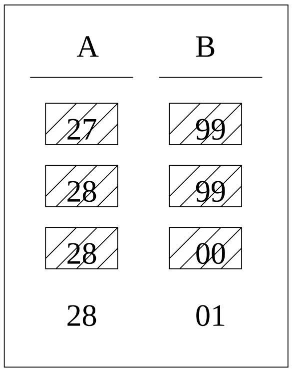
Lecture 26

In this lecture we look at electronic voting protocols and the use of Zero Knowledge proofs in protocol design.

1 Voting Protocols

1.1 A Toy Example

Suppose we wanted to vote for future lecture topics in this class. One sheet of paper with 2 columns, or more if there are more choices, can be passed around the class. Each column starts with a random number modulus 100. As the sheet is passed around the class, each student increments the number under the column of their choice and erases the previous value. The value in the other column simply gets copied over and the previous entry is erased as well. The initial values effectively act as a one time pad and a trusted party can easily figure out the results of the election as it knows the initial values for each column.



There are a number of obvious problems with this scheme:

- The integrity of the vote can easily be compromised. Any voter can easily increment the column counter by an arbitrary value. A smarter malicious voter could increment the count for one column by 2 while decrementing the other by one to keep the correct total number of votes cast.
- The protocol has full trust in the administrating agent to calculate the votes correctly.
- Voter privacy is compromised as colluding voters can learn how other votes. Two colluding voters can easily learn the choice of a voter sitting between them.
- If the trusted agent can observe each communication, without observing the voting process itself, it can easily figure out how each voter voted by simply observing the intermediate change in the tally.
- A denial of service attack is extremely easy to mount as a malicious voter can simply refuse to pass the sheet around once it gets its hand on it and stall the entire election.

1.2 Fixing the Protocol

To obtain the privacy property and to guard against right and left neighbor collusion an additively homomorphic encryption scheme can be used. A previous count $E(r')$ can either be reblinded with $E(0)$ to get a new encryption of $E(r')$ or incremented with $E(1)$.

$$\begin{array}{cc} E(r) & E(r') \\ E(r+1) = E(r) \cdot E(1) & E(r+0) = E(r) \cdot E(0) \end{array}$$

This protocol can be implemented in a computer system by using a publicly accessible append-only bulletin board. Assuming an encryption scheme that is semantically secure, the voters can't tell what the previous values were and can't tell which way the new vote went. To guard against multiple voting, we want to prevent voters from writing multiple rows to the bulletin board. To achieve this, the protocol can require for the voter identity to be provided with every vote entry. Using public key encryption, a voter can sign their votes with their private key and multiple voting entries can easily be detected.

To make sure that each vote increments only one column and exactly by 1, in going from $c \to d$ in one column and $c' \to d'$ in the other, we want to prove that

$$(D(d) = D(c) \wedge D(d') = D(c) + 1) \vee (D(d) = D(c) + 1 \wedge D(d') = D(c))$$

where $D(\cdot)$ is the decryption algorithm for the encryption scheme used on the bulletin board. Since given the private key, this boolean expression can verified in polynomial time by simply decrypting the four values, the problem is in NP and there is a Zero Knowledge proof that we can use to verify that each voter increments the tally by only one vote without learning what the vote actually was. The Fiat-Shamir heuristic can be used to transform the interactive zero knowledge proof into a non-interactive one. This provides the protocol with the universal verifiability property, where it becomes possible for anyone to use the bulletin board to check the proof.

With a shared bulletin board, the type of denial of service attack discussed above is not possible as the progress of the election does not depend on any one voter and no one can hold anything up. The solution to not having to rely on one possibly untrusted agent is to have k trustees such that not all of them would collude.

This problem was worked on in the late 80's and early 90's and was though to be solved. There are however additional properties that are desired in a voting protocol which are not addressed here:

- Voters should not be able to prove how they voted to anyone else. This avoid the problem of vote buying, where a voter can be swayed to vote a certain way, or the problem of coercion where a voter can be forced to prove that he voted as he was told.
- If the voter needs to rely on a computer (e.g., to encrypt various values), the voter needs some way to gain trust in the correct operation of the computer. For instance, the computer might prove somehow to the voter that it performed all calculations correctly and as required by the protocol.

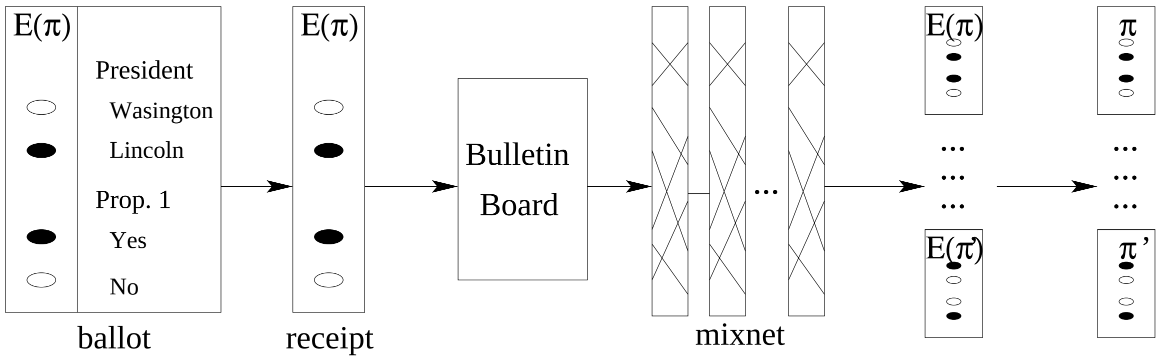


Figure 1: Chaum's voting protocol

1.3 Chaum's Paper Ballot Scheme

The voting scheme is based on paper ballots and gives assurance that counting of votes is done correctly. Each voter can verify that their vote was included in the count and any one can verify that the voters' ballots were counted correctly, providing universal verifiability. Each ballot is split in two parts, with the runoffs and the candidate choices on the right, and the bullets to indicate a voter's choice on the left.

Every ballot is unique, as a permutation π is used to permute the order of candidate names. The permutation is placed on the left side of the ballot in encrypted form. Just looking at the left side of the ballot, one cannot tell what the vote was. The full ballot goes into the ballot box and the left side of the ballot is copied with a xerox machine, the image is saved and a copy is given to the voter as their receipt. The voter can visually verify that the copy was made correctly. The electronic image of the receipt is posted to an append-only world-readable bulletin board. The permutation π appears on the ballot in encrypted form and is only meaningful to trustees.

The protocol needs to be able to verify the outcome of the election without revealing the privacy of the voters. A mixnet is used to reorder and reblind the receipts appearing on the bulletin board before the actual votes are revealed. The mixing party can provide a zero knowledge proof that verifies that there was no tampering with the ballots while shuffling the ballots. If the mixer is honest then it also offers unlinkable shuffling. Threshold decryption is used to decrypt the permutations used with each ballot so that the votes can be counted. Threshold decryption also give a zero knowledge proof of the fact that the decryption was done correctly. The final properties of the protocol are such that:

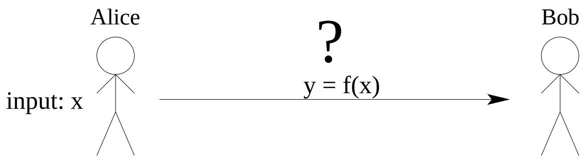
1. Anyone can check correct tally of the votes as they are posted on the bulletin board by checking the shuffling and decryption proofs

2. Any voter can make sure that their receipt appears on the bulletin board correctly and therefore counted in the election

One issue is that the ballots in their original form contain the permutation in encrypted form and a voter cannot tell if the permutation on the ballot actually matched the order of candidates that is listed. The election officials should be able to prove that the original ballots are correct and that the permutations do match the printed order without revealing what the permutation actually is. One way to solve this problem is to have the election officials print twice as many ballots as needed. Each each voter would take two ballots at random, use one and keep the other, and have the officials prove that the permutation is correct on the unused ballot.

2 Protocol Design Principles

The scenario under scrutiny is one in which two parties are supposed to follow a protocol and one party wonders if the protocol was actually followed. More specifically, suppose that Alice knows a value x and is supposed to compute a function f on that input and send the result $y = f(x)$ to Bob. Bob sees some value y sent by Alice and wonders if it's really $f(x)$. The question here is how can Bob tell if Alice followed the prescribed protocol and how can he detect if Alice deviated from it?



2.1 Public Inputs

The simplest case to consider is when both the input x is public and the function f is deterministic and public as well. In this case Bob can compute f(x) and compare it to the value y that Alice sent. In fact, in this scenario Alice doesn't even need to send y to Bob as Bob can simply compute f(x) himself.

2.2 Randomized Input

Now, x is public and f is public and deterministic but takes two inputs, x and r, which is chosen at random by Alice.

$$\begin{array}{ccc} \underline{\mathit{Alice}} & & \underline{\mathit{Bob}} \\ r \xleftarrow{\$} R & & \\ & \xrightarrow{y=f(x,r)} & \end{array}$$

Two issues need to be dealt with. First, we want r to stay private; it might be an input to a public-key generation procedure. Second, Bob may be relying on the assumption that r is picked at random and is not specifically chosen. In the following coin flipping protocol Bob makes sure that r is indeed chosen at random by forcing Alice to use randomness supplied by Bob and then making Alice prove that she actually used it. The downside of this protocol is that in the process of proving that Alice used Bob's random input she has to reveal the value of r, which is undesired.

$$
\begin{array}{llcl}
 & \underline{Alice} & & \underline{Bob} \\
1: & r_a \xleftarrow{\$} R & \xrightarrow{c=commit(r_a)} & \\
2: & & \xleftarrow{r_b} & r_b \xleftarrow{\$} R \\
3: & r = r_a \oplus r_b & \xrightarrow{r,open(r_a)} & \\
4: & & \xrightarrow{y=f(x,r)} &
\end{array}
$$

In the ideal situation, Alice would want to prove to Bob that it used the value r_a that it committed to in step 1 together with Bob's value r_b to generate $y = f(x, r_a \oplus r_b)$ without Bob gaining any knowledge about the actual value of r_a. This would effectively eliminate step 3 in the protocol above. In step 4, together with the value of y, Alice presents a Zero Knowledge proof that

$$ZK(\exists w.verify(w, c) = 1 \wedge f(x, w \oplus r_b) = y)$$

and since there is a polynomial time verification algorithm which can take the witness w and make sure that it could have played the role of r_a, the language is in NP and there is a ZK proof. Such protocols can be used to prove that proper encryption was performed or that proper rebinding has taken place.

2.3 One Secret Input

In this scenario, x and f are public, but Alice has a secret input s and is supposed to send $y = f(x, s)$ to Bob. In the protocol below, Alice first commits to the secret value s, receives the value x from Bob, and then sends y along with an interactive zero knowledge proof that y was obtained by using the value committed to in step 1.

$$
\begin{array}{llcl}
 & \underline{Alice} & & \underline{Bob} \\
1: & x, f, s & \xrightarrow{c=commit(s)} & x, f \\
2: & & \xleftarrow{x} & \\
3: & & \xrightarrow{y=f(x,s)} & \\
 & & ZKIP(\ldots) &
\end{array}
$$

2.4 Two Secret Inputs

Another protocol is one in which a function of two inputs is known to both parties, but Alice known a private value x and Bob known a private value y and they both want to know the value of $f(x, y)$. This is a harder problem but it can be solved using secure circuit evaluation [?]. In such a protocol Alice knows some circuit c and Bob knows an input to the circuit y and they both would like to learn what the output of the circuit is on the input data without revealing their respective private inputs.