6.897: Advanced Topics in Cryptography April 29, 2004

Lecture 23: Chaum's Election Scheme

Lecturer: Ron Rivest *Scribed by: Yoav Yerushalmi*

1 Lecture Overview

In the previous lecture, we discussed a Rivest-devised variation on Chaum's election scheme. We also introduced the notion of visual cryptography, and presented slides to go with the topic. In this lecture, we will finish describing the variant, as well as delve into Chaum's original scheme.

Recall that the purpose behind these schemes is to have an election that can be conducted electronically, but that can be verified at every step. We wanted to have a paper trail, but one that can't be used as a receipt.

2 Conclusion of Ron's Variant on Chaum's Scheme

Recall that a vote for a voter named V_i in the variant is constructed via a chain $B_i \to C_i \to D_i$, where B_i is the original ballot, C_i is the first ElGamal encryption $(g^{r_i}, B_i y^{r_i})$, and D_i is the re-encryption $(g^{r_i+s_i}, B_i y^{r_i+s_i})$.

A complete vote is the path from B_i to D_i, but we only let the user take home $B_i \to C_i$ or $C_i \to D_i$. He picks one of them randomly, and so the machine has a 1/2 chance of being caught if it is cheating. The receipt is $R_i = (V_i, D_i)$, which consists of his name and his encrypted ballot, plus signatures on R_i: $\sigma_I(R_i), \sigma_F(R_i)$, where σ_I is generated by the machine for each R_i it outputs (and there will be more than one such if the voter changes his mind before casting his vote), and σ_F is generated after the user makes his decision to cast his vote (this is mostly for tracking when/where cheating may have occurred).

The compliance evidence presented to the user is:

$$\begin{aligned} S_i &= (B_i, C_i, r_i), \sigma_I(S_i) \\ T_i &= (C_i, D_i, s_i), \sigma_I(T_i) \end{aligned}$$

The user then casts his vote, and receives $\sigma_F(R_i)$, which is also posted on some giant bulletin board.

The user gets to take with him $R_i, \sigma_I(R_i), \sigma_F(R_i)$, and either $(S_i, \sigma_I(S_i))$ or $(T_i, \sigma_I(T_i))$. Guards at the polling station ensure the other one is destroyed. At home, (or via some public service provided by watchdog groups) the user will verify that the parts he has are consistent. If they are not, he will get to complain and present what evidence he has. Ideally, if the machine cheats significantly, enough claims of error will be made that a new vote will take place. If only one complaint is registered, unless the signatures are all valid, it is more likely that the user is cheating.

The simplified scheme introduced us to the basic concepts behind Chaum's scheme:

- Users can provide **cut and choose** functionality (acting as an unpredictable source of randomness).
- We can allow users to have multiple votes generated, but only the final one counts (in case they need to present false receipts to a malicious third party.

There is some debate however, whether allowing users to choose which receipt to take home is a good plan. One receipt clearly contains their ballot, whereas the other looks like random garbage. In that scenario, it may be the case that most users will choose the non-garbage half. To resolve this, we introduce the final trick: using visual crypto.

We assume that the machine is not colluding with the voter. Instead of generating the chain $B_i \rightarrow C_i \rightarrow D_i$, we generate two chains which xor to the original values. Note that one of the chains is just a random choice of values:

$$\begin{array}{ccc} B_i' & \xrightarrow{r_i'} C_i' & \xrightarrow{s_i'} D_i' \\ \oplus & & \\ B_i'' & \xrightarrow{r_i''} C_i'' & \xrightarrow{s_i''} D_i'' \\ = & & \\ B_i & & \end{array}$$

They are then printed on two transparencies so that when overlayed, the user can verify that the B_i so formed as the xor of B_i' and B_i'' that he sees does indeed correspond to his intended choice. He then takes home one of the two transparencies containing either (B_i', r_i', s_i'') or (B_i'', r_i'', s_i'), as well as the original values $R_i = (V_i, D_i', D_i'')$ and $\hat{C}_i = (C_i', C_i'')$.

The D_i's are kept together through the mixnet, but at the reveal phase, we reveal only B_i, not the two halves (otherwise there would be receipts generated). Sadly, proving the decryption was done correctly in this scheme seems hard.

The above is all just a warm-up for Chaum's actual proposal.

3 Chaum's Scheme

Chaum's original scheme is fairly similar to the variant above, but somewhat harder to follow.

Each party gets a value q which is a serial number associated with their vote. They generate a ballot image $B \in \{0,1\}^{m \times m}$.

To place a vote, the machine generates two transparencies, but we want to ensure that both transparencies contain enough information about the ballot so that the right parties can figure out the vote from only one transparency. There will therefore be a white part and a red part generated, and we will view the transparencies as a top or bottom half.

The white part (W) is seemingly random data (we will talk about how it is generated). We then compute the red $R = W \oplus B$. Now we can generate the top and bottom halves and ensure both contain information about B. Let $B = (B^t, B^b)$ where each has enough redundancy so that B can be fully constructed from either. The values L_t and L_b are then the two layers (i.e., what is actually printed on the top and bottom transparencies).

$$\begin{aligned} R^t &= W^b \oplus B^t \\ R^b &= W^t \oplus B^b \\ \\ L^t &= R^t, W^t \\ L^b &= W^b, R^b \end{aligned}$$

Note that when the two layers are placed on top of each other, the ballot image is visible. If the voter approves of the ballot, he will select a layer x at random. The machine will append $q, s^x(q)$ (where $s^x()$ is a signature and also is the seed used to generate W^x). Finally, it will append a pre-mixed "onion" (D^t_k, D^b_k). One transparency is destroyed, and the other is submitted/published.

The W's are generated in a way that allows us to use a decryption mixnet to reveal the ballots.

$$\begin{tikzcd}
& S^x(q) \arrow[dl, dash, "h(x,1)"'] \arrow[d, dash, "h(x,2)"] \arrow[dr, dash, "h(x,k)"] & \\
d_1^{x'} \arrow[d, dash, "h'"'] & d_2^{x'} \arrow[d, dash, "h'"'] & d_k^{x'} \arrow[d, dash, "h'"'] \\
d_1^x & d_2^x & d_k^x
\end{tikzcd}$$

We then set $W^x = d_1^x \oplus d_2^x \oplus \ldots \oplus d_k^x$.

Each $d_i^{x'}$ is used in one of the decryption mixes in that order. We refer to the intermediate values as "dolls", and generate them this way:

$$\begin{aligned} D_0 &= 0 \\ D_l &= e_l(d_l^{x'} || D_{l-1}) \\ \\ B_k^x &= R^x \\ B_l^x &= B_{l+1}^x \oplus d_l^x \\ B_0^x &= B^x \end{aligned}$$

Each mixnet machine in turn receives (D_l, B_l) and can remove one layer of the "onion" by revealing the local $d_l^{x'}$ through its private key. It can then hash the value to get d_l^x which it can use to help recover B^x. It will spit out (D_{l-1}, B_{l-1}). As always, it must additionally provably shuffle the results using techniques shown in previous lectures.

Next lecture we will discuss Neff's election scheme, and see how it compares to these.