6.879 Special Topics in Cryptography Instructors: Ran Canetti

Lecture 19: Verifiable Mix-Net Voting

April 15, 2004 *Scribe: Susan Hohenberger*

In the last lecture, we described two types of mix-net voting protocols: *decryption mix-nets* and *re-encryption mix-nets*. Today, we direct our focus to El Gamal based re-encryption mix-nets. The malleability of El Gamal encryptions, a drawback in many applications, is beneficially used to re-encrypt each ciphertext with new randomness without knowledge of the underlying ballot contents. Further, the output of each El Gamal re-encryption mix-net can be publically verified using a well-known protocol for proving the equivalence of two discrete logarithms [CP92] and a few other tricks. Today, we begin to show how a concerned citizen can verify the output of a mix-net server. We also discuss some ways in which a malicious voter might attempt to "cheat the system" and provide some techniques for discouraging this undesirable behavior.

1 The Challenges of Verifiable Mix-Net Voting

Let us briefly recall the key steps in our El Gamal, verifiable mix-net voting protocol, as illustrated in Figure 1.

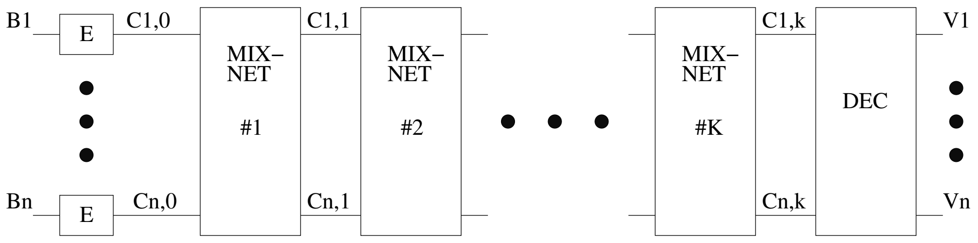

Figure 1: A Mix-Net Voting Protocol

First, a group of respectable, but mutually distrusting parties (such as the Republican Party, the ACLU, an election official, etc.) jointly carry-on a secure computation protocol to generate an El Gamal keypair of the form (g^x, x), where the public key g^x is published and the secret key x is known to no one, but can be recreated from a threshold of shares distributed among the parties. Next, ballots $B_1 \ldots B_n$ are generated by n different voters and encrypted, under g^x, to the ciphertexts $C_{1,0} \ldots C_{n,0}$ with the help of some (hopefully trustworthy) computing resources. Then the ciphertexts are sent through k mix-net servers, presumably operated by a host of respectable parties with competing interests, such as the Republican, Democratic, and Green Parties. All ciphertexts are posted to a public bulletin board. Once the outputs of the mix-servers are verified to be correct, a threshold of the parties jointly agree to decrypt the ballots and count the vote.

2 Keeping the Voter Honest

Any secure voting protocol must defend itself against a host of adversaries. The voter herself may attempt to "cheat the system"; that is, behave in ways that defeat the ideals of a voting democracy. One such example, that we discuss in detail in the next lecture, is vote selling. Any valid voting scheme should not allow a person to *prove* how they voted to a third party. Today, we look at two other problems that could arise during the production of the initial encrypted ballots $C_{1,0} \ldots C_{n,0}$.

1. **Canceling a vote.**

   Suppose Bob cares nothing about the election, but wishes to thwart Alice's attempt to express her opinion ... whatever that opinion is. We say that Bob *cancels* Alice's vote if, after seeing her encrypted vote $C_{a,0}$ and *without* knowledge of how she voted, Bob can generate a ciphertext $C_{b,0}$ that votes against Alice's preference. For example, if Alice votes yes on issue 23, then we want to prevent Bob from creating an encrypted ballot with the opposite vote (i.e., no on issue 23) without knowing how either he or Alice voted.

2. **Copy-cat voting.** A similar scenario that we also want to prevent is *vote copying.* If Alice tells Bob she is voting yes on issue 23 and he decides to vote the way she does, that is fine. But if Alice chooses not to tell Bob how she voted on an issue, then he should not be able to simply "copy" her vote. (It is not hard to imagine various kinds of voter coercion based on copying.)

Both vote canceling and vote copying can be prevented if Bob is forced to prove knowledge of how he is voting; that is, when Bob presents his encrypted ballot $C_{b,0}$, he must be prepared to give a zero-knowledge proof of knowledge of the plaintext contents of the ballot.

2.1 Proving Knowledge of Plaintext

Given an El Gamal ciphertext $C = (\alpha, \beta) = (g^t, my^t)$ under the public key y, a voter can prove knowledge of m (i.e., how they voted) by instead proving knowledge of t. The justification is that the voter could obtain m from C using t.[1] An efficient, 3-round protocol for proving knowledge of t, assuming that g^t is public, is as follows:

$$\begin{array}{ccc}
P & & V \\
s \in \mathbb{Z}_q & \xrightarrow[A = g^s]{} & \\
 & \xleftarrow[c]{} & c \in \mathbb{Z}_q \\
 & \xrightarrow[r = s + ct]{} & \text{Check that: } g^r \overset{?}{=} Ag^{tc}
\end{array}$$

[1] Shafi Goldwasser and Ran Canetti suggested simply using CCA2-secure Cramer-Shoup [CS03] instead of El Gamal encryption for the ballots, since its non-malleability prevents an adversary from getting a related message in a different ciphertext (assuming that we disallow the *exact* same ciphertext for two voters). This idea was debated for a while, until it was given up on account of that fact that the Cramer-Shoup secret key might be necessary to verify the output of the mix-nets. One of the core properties that we want from a voting protocol is public verification.

Here the prover P is the voter, possibly working together with the trusted encryption software E in Figure 1, while the verifier V might be an election official.

2.2 Honest Verifier Zero Knowledge

The protocol in the previous section is an honest verifier zero knowledge (HVZK) proof of knowledge protocol. An HVZK protocol has a special soundness property: if the verifier gets to see two transcripts that use that same initial commitment from the prover $A = g^s$ where the verifier is allowed to use two different challenges c_1, c_2 and receive the responses $r_1 = s + c_1t, r_2 = s + c_2t$, then the verifier may compute t. (Since the verifier knows r_1, r_2, c_1, c_2, he can solve for s and t.) For more examples of HVZK applications, see the work of Jakobsson and Juels [JJ99] and Cramer, Damgard and Schoenmakers [CDS94].

3 Keeping the Mix-Server Honest

A re-encryption mix-server can behave maliciously in two ways: (1) it can attempt to alter the vote count, and (2) it can attempt to destroy voter privacy. In this section, we discuss how to defeat the first adversarial behavior by forcing the mix-server to provide proof that it did not alter the vote count. Destroying voter privacy, by leaking information about the permutation used, is beyond our mathematical ability to solve. It is impossible for an election official to verify that a mix-server did *not* sell its permuation information to the mafia. Thus, the accuracy of the vote count is computationally secure (DL hard so that mix-server can't fake proofs) while the voters' privacy depends on at least one of the mix-servers being honest (i.e., keeping its permutation information secret).

Each mix-server takes in k El Gamal ciphertexts $C_i = (\alpha_i, \beta_i)$, re-encrypts them, randomly permutes them, and outputs this new sequence of ciphertexts. To prove that it did not alter the vote count, each mix-server must prove that each ballot in the input appears in the output.

More formally, each mix-server must prove the following NP statement:

$$\begin{aligned}
&\exists \pi \text{ on } \{1, \ldots, n\} \text{ nodes },\\
&\quad \exists \text{ elements } s_1, \ldots, s_n,\\
&\qquad \forall i \in \{1, \ldots, n\},\\
&\qquad\qquad C_{i,0} = (\alpha_{i,0}, \beta_{i,0}) \text{ and } C_{\pi(i),1} = (\alpha_{\pi(i),1}, \beta_{\pi(i),1}),\\
&\qquad\qquad \text{where } \alpha_{\pi(i),1} = \alpha_{i,0} g^{s_i},\ \beta_{\pi(i),1} = \beta_{i,0} y^{s_i}.
\end{aligned}$$

Since it is an NP statement, it is clear that we can prove this in zero-knowledge [BGG$^+$88], the question is: how fast? We first consider the following subroutine.

3.1 The Chaum-Pedersen Protocol

This section was editted from a gracious donation by Yael Kalai.

First, we consider the simpler task of merely proving that one ciphertext is a re-encryption of another. Let $c_1 = (\alpha_1, \beta_1) = (g^t, m_1 y^t)$ and $c_2 = (\alpha_2, \beta_2) = (g^u, m_2 y^u)$

be any two ciphertexts. Note that c_2 is a re-encryption of c_1 if and only if c_1 and c_2 are both encryptions of the same message. Let

$$(a_1, a_2, b_1, b_2) = (g, y, \frac{\alpha_2}{\alpha_1}, \frac{\beta_2}{\beta_1}) = (g, y, g^{u-t}, \frac{m_2}{m_1} y^{u-t}).$$

Claim 1 *The ciphertext c_2 is a re-encryption of c_1 if and only if $\log_{a_1}(b_1) = \log_{a_2}(b_2)$.*

Proof 1 *First, we consider the forward implication. If c_2 is a re-encryption of c_1, then $(a_1, a_2, b_1, b_2) = (g, y, g^{u-t}, \frac{m_2}{m_1} y^{u-t}) = (g, y, g^{u-t}, y^{u-t})$ since $m_1 = m_2$. Thus, we see that $\log_g(g^{u-t}) = \log_y(y^{u-t}) = u - t$.*

We consider the reverse implication. If $\log_{a_1}(b_1) = \log_{a_2}(b_2)$, then $c_2 = (\alpha_1 g^v, \beta_1 y^v)$ for some $v \in \mathbb{Z}_q$, but this term can be factored out of the ciphertext to reveal $c_2 = c_1(g^v, y^v)$, where (g^v, y^v) is simply an encryption of 1!

Thus, proving that c_2 is a re-encryption of c_1 boils down to proving that $\log_{a_1}(b_1) = \log_{a_2}(b_2)$ or in other words that (a_1, a_2, b_1, b_2) is a DDH tuple. We can do this by running the Chaum-Pederson protocol [CP92] for proving that a tuple $(g, y, w, u) = (g, g^x, g^r, g^{rx})$ is a DDH tuple. We now describe that procotol:

$$\begin{array}{ccc}
P & & V \\
s \in \mathbb{Z}_q & \xrightarrow[(a,b) = (g^s, w^s)]{} & \\
 & \xleftarrow[c]{} & c \in \mathbb{Z}_q \\
 & \xrightarrow[t = s + cr]{} & \text{accept if and only if} \\
 & & g^t = aw^c \wedge y^t = bu^c
\end{array}$$

Here, all that the prover must know is the *re-randomization factor* r; not the message itself! The above protocol is an honest verifier zero-knowledge proof-of-knowledge (of x) protocol.

3.2 Considering a $n \times n$ Mix

Now that we know how to show that an El Gamal ciphertext c_2 is a re-encryption of a ciphertext c_1, we want to use this to build a $n \times n$ mix. That is, we want to prove that $(c_1,' \ldots, c_n')$ are re-encryptions of $(c_1, \ldots, c_n)$ *without* revealing the actual correspondence.

One naive approach is to have the kth mix-server prove the following NP statement in zero-knowledge:

$$\begin{array}{l}
\forall i \ \exists j \text{ such that } C_{i,k} \approx C_{j,k+1} \text{ and} \\
\forall j \ \exists i \text{ such that } C_{i,k} \approx C_{j,k+1}
\end{array}$$

Unfortunately, an $n \times n$ mix-server could prove this statement is true and still be cheating. To see this, consider the case where $n = 3$. Suppose, input to the mix-server, were two votes for Donald Duck and one vote for Minnie Mouse. The output of a malicious mix-server could be one vote for Donald Duck and two votes for Minnie Mouse – and the above statement would still be true!

However a quick inspection of the possibilities shows that the above statement is sufficient for a simple 2×2 mix. We will therefore build a general-purpose $n \times n$ mix out of 2 $\times$ 2 mixes – and we will see how to do this next class.

References

[BGG$^+$88] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Hastad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything Provable is Provable in Zero-Knowledge. In *Advances in Cryptology - Crypto '88*, pages 37–56, 1988.

[CDS94] Ronald Cramer, Ivan Damgard, and Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Advances in Cryptology - Crypto '94*, pages 174–187, 1994.

[CP92] David Chaum and T.P. Pedersen. Wallet Databases with Observers. In *Advances in Cryptology - Crypto '92*, pages 89–105, 1992.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal of Computing*, 2003. To appear. Available at http://www.shoup.net/papers.

[JJ99] Markus Jakobsson and Ari Juels. Millimix: Mixing in Small Batches. Technical report, 1999.