6.897: Advanced Topics in Cryptography Apr 22, 2004

Lecture 21: Neff (VoteHere) Voting Scheme

Lecturer: Ron Rivest *Scribed by: Chris Peikert*

Topics for this lecture:

- A comment on Neff's mix-net
- Batch verification
- The Boneh-Golle scheme
- Randomized partial checking (RPC) of mix-nets

1 Comment on Neff's mix-net

Recall the novel trick used by Neff: each mix server is also a decryption server (in an "onion" architecture). During the mixing, an ElGamal ciphertext $(g^r, m \cdot y^r)$ is re-randomized by each server by choosing a random and secret c, and producing $(g^{rc}, m^c \cdot y^{rc})$. In addition, the pair (g, g^c) is published by the server.

Upon decryption, the server takes cth roots of each ciphertext and proves correctness by showing equality of discrete logs (e.g., via Chaum-Pedersen) using the published (g, g^c) pair.

2 Batch verification

Consider the following abstract problem (outside the context of voting): given g and n pairs (x_i, y_i), verify that each $y_i = g^{x_i}$.

The naive solution is to just perform n exponentiations and compare the results with the y_is. But consider the case that $n = 2$: we can save an exponentiation by checking if $(x_1 + x_2, y_1 \cdot y_2)$ is a good pair. Certainly if the two original pairs are good, so is the third.

The converse isn't true in general, but with randomization we can get a good chance of detecting a bad set of pairs. The general test is to pick a random subset $S \subseteq [n]$, combine, and check. I.e., check that

$$\left(\sum_{i \in S} x_i, \prod_{i \in S} y_i \right)$$

is a good pair, using only one exponentiation. See [BGR98] for information about the batch verification techniques in today's lecture.

Theorem 1 *If the original set of pairs is invalid, then the above check detects this fact with probability at least* $1/2$.

It's easy to see that the detection probability can't be any better than 1/2: if there is only one bad pair in the set, it will be excluded half the time, in which case the test passes.

Instead of proving Theorem 1, we will introduce a more general test and bound its error probability.

The generalized test is the following: instead of choosing a single random bit of pair i (i.e., "include i in S" or not), pick a random s_i from some small range $\{0, \ldots, 2^{t-1}\}$. Then check that

$$\left(\sum_{i \in [n]} s_i x_i, \prod_{i \in [n]} y_i^{s_i} \right)$$

is a valid pair.

In terms of computation, this test only takes about $1.5t$ multiplications (to compute $y_i^{s_i}$) per pair, but only one general exponentiation.

Notice that this test has some issues in the application to mix-nets: for the first test, the mix only needs to reveal the set of outgoing ciphertexts corresponding to the random set S. This compromises voter privacy a little bit, but not too much. In the second test, the mix must reveal the set of outgoing ciphertexts corresponding to each of the 2^t possible values of s_i. This leaks significantly more information about how votes are permuted through the mix-nets, though it may not be entirely fatal.

Theorem 2 *The above test will accept invalid input with probability at most* 2^{-t}.

Proof: If the set is invalid, let's isolate one invalid $(\bar{x}, \bar{y})$ pair, and let $\bar{s}$ be the corresponding random coefficient. Then the test checks whether $(\bar{s}\bar{x} + \sum s_j x_j, \bar{y}^{\bar{s}} \prod y_j^{s_j})$ is a valid pair. Let $\bar{z} \neq \bar{x}$ be the discrete log of $\bar{y}$, base g. Then the discrete log of the right half of the pair is $\bar{s}\bar{z} + c$, where c is some constant independent of $\bar{s}$. In order for the test to pass, we need $\bar{s}\bar{x} + \sum s_j x_j = \bar{s}\bar{z} + c$, where the summation is also independent of $\bar{s}$. Because g has prime order and $\bar{x} \neq \bar{z}$, there is only one value of $\bar{s}$ that causes the test to pass, and it is chosen with probability 2^{-t}. □

Now let's see how these tests are applicable to mix-nets.

3 The Boneh-Golle approach to mix-nets

We want to efficiently verify that a mix has operated properly, i.e. that it has permuted its inputs and re-encrypted them using an additive random pad in the exponent.

Here is the idea: pick a random subset S of the inputs, and get the input ciphertexts c_i for each $i \in S$. Combine all these inputs, via component-wise multiplication, to get $\hat{c}$, a "meta-input" which is a ciphertext of $\hat{m} = \prod_{i \in S} m_i$.

Now, challenge the mix server to produce a corresponding *set* S', of the same size as S, such that the "meta-output" (defined similarly) encrypts the same message $\hat{m}$. (Note that the set S' should be permuted so as not to reveal the exactly correspondences between input and output ciphertexts.) The proof, as always, is done by showing equality of discrete logs.

Let's be careful: there is a striking difference between what is *true* about the meta-output, and what is *provable* by the mix sever. For example, it is possible that the mix

changes some votes (turns 10 votes for Carter and 10 votes for Reagan into 11 votes for Carter and 9 votes for Reagan), yet can almost always find output ciphertexts such that the meta-output encrypts the same meta-message as the meta-input. (In our example, the only problematic challenge set S is the one exactly corresponding to the 10 votes for Reagan).

The informal theorem is that it is *hard to prove* that invalid mixing is correct. More formally:

Theorem 3 (Thm 7.1 in Boneh-Golle) *If the mix server operated incorrectly, and if the mix can satisfy a random challenge with probability at least* $3/8 - \epsilon$*, then discrete log can be solved in polynomial time.*

Notice that we lose some privacy with each repetition of the proof: with α challenges, we can narrow down an input ciphertext's corresponding to among about $n/2^\alpha$ of the outputs. Perhaps the serial nature of the mixes may better hide the overall input-output correspondences, but this seems hard to analyze.

Here is an idea for a fix: apply the idea behind the 2×2 mix. Choose a set S and combine the ciphertexts into 2 meta-inputs (corresponding to S and $\bar{S}$) and 2 meta-outputs (corresponding to T and $\bar{T}$), and show that all four sets are the same size and that the meta-outputs encrypt the same values as the meta-inputs (but without giving the correspondence).

Research Question 1 *Is this scheme still secure? The proof of Theorem 3 depends on the use of a* random *subset S, but the above technique only works if $|S|$ is exactly $n/2$ (otherwise it is clear how the meta-outputs correspond to the meta-inputs).*

4 Randomized partial checking (RPC)

Here is a quick summary of a scheme using ideas from [JJR02] and Chaum: each organization that does mixing actually controls two consecutive mixes, and does normal mixing for both. Challenges are issued on the *middle*; that is, a random subset S of the middle ciphertexts is chosen. For each ciphertext in S, its corresponding input ciphertext is identified and proven to be correct. For each ciphertext in $\bar{S}$, its corresponding output ciphertext is identified and proven to be correct.

There is some loss of privacy: across two consecutive mixes, one can learn that an input ciphertext corresponds to one of $n/2$ outputs ciphertexts.

There is a novel soundness property: when cheating, the chance of being caught is *exponential* (base 1/2) in the number of votes that the mix attempts to change! This seems reasonable for large elections, though not for ones that may decided by only a few votes. (In these cases, we can invoke one of the slower proof techniques on the original data.)

For efficiency, we can do batch verification of the ciphertext correspondences. This causes no loss in privacy, but it affects the soundness.

References

[BGR98] M. Bellare, J. Garay, and T. Rabin. Batch verification with applications to cryptography and checking. In *Proceedings LATIN '98*, pages 170–191, 1998.

[JJR02] M. Jakobsson, A. Juels, and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In D. Boneh, editor, *USENIX Security 2002*, pages 339–353, 2002.