6.879 Special Topics in Cryptography Instructors: Ran Canetti and Ron Rivest

Lecture 20: Verifiable Mix-Nets

April 16, 2004 *Scribe: Matt Lepinski*

1 Two-By-Two Verifiable Mixes

Before solving the general case of verifiably mixing n ciphertexts, we first consider the case of a verifiable two-by-two with two incoming ciphertexts, C_1 and C_2 and two outgoing ciphertexts C_1' and C_2'. In this case, what the mixer must prove is that:[1]

$$(C_1 \approx C_1') \wedge (C_2 \approx C_2') \vee (C_1 \approx C_1') \wedge (C_2 \approx C_1')$$

This is equivalent to proving:

$$[(C_1 \approx C_1') \vee (C_1 \approx C_2')] \wedge [(C_2 \approx C_1') \vee (C_2 \approx C_2')] \wedge [(C_1' \approx C_1) \vee (C_1' \approx C_2)] \wedge [(C_2' \approx C_1) \vee (C_2' \approx C_2)]$$

Observed that the above expression is a conjunction of four disjunctions. We will give a protocol which allows the prover to prove a single disjunction. The prover can then prove the entire expression by separately proving that each of the four disjunctions is true.

First we recall from last lecture the Chaum-Pedersen honest zero knowledge protocol for proving that two El Gamal ciphertexts, $C_1 = (\alpha_1, \beta_1) = (g^t, m_1 \cdot y^t)$ and $C_1' = (\alpha_1', \beta_1') = (g^u, m_1' \cdot y^u)$ have the same plaintext (where the prover knows the re-encryption factor, $v = u - t$).[2] Let (a_1, a_2, b_1, b_2) be the quadruple $(g, y, (\alpha_1'/\alpha_1), (\beta_1'/\beta_1)) = (g, y, g^v, (m_1'/m_1) \cdot y^v)$. Then $m_1 = m_2$ if and only if $\log_{a_1}(b_1) = \log_{a_2}(b_2) = v$. (Proof left as an easy exercise.) To prove equality, use the following protocol:

	P	V
randomly select s from $\mathbb{Z}_q^*$:	$\bar{A} = (\bar{A}_1, \bar{A}_2) = (a_1^s, a_2^s) \longrightarrow$	
	$\longleftarrow c$	: randomly select c from $\mathbb{Z}_q^*$
$r = s + c * v$:	$r \longrightarrow$	
		Accept if $a_1^r = A_1 b_1^c$ and $a_2^r = A_2 b_2^c$

(This is really two parallel instances of the previous Chaum-Pedersen protocol, sharing r and c. Although we don't need this fact here, this protocol could be showing equality of logarithms in two distinct groups.) The protocol is honest-verifier zero-knowledge; it is also a proof of knowledge of v. Moreover it is "special HVZK": given any specific c, one can pick $\bar{A}$ and r to match the conditional distribution of the transcripts, given c.

We now present an honest zero-knowledge protocol for proving the disjunction $[(C_1 \approx C_1') \vee (C_1 \approx C_2')]$ (where the prover knows either the first re-encryption factor r_1 or the

[1] Recall that we denote the relation "C has the same plaintext as D" by writing $C \approx D$.

[2] Here we assume that g is a publically known generator of Z_q^*.

second re-encryption factor r_2):[3] This protocol is derived from the paper by Cramer et al. [CDS94].

$$\begin{array}{ccl} P & & V \\ & \bar{A}_1, \bar{A}_2 \longrightarrow & \\ & \longleftarrow c & : \text{randomly select } c \\ & r_1, r_2, c_1, c_2 \longrightarrow & \end{array}$$

The verifier accepts if (1) the Chaum-Pedersen verifier would accept the triple $(\bar{A}_1, c_1, r_1)$ with ciphertexts C_1 and C_1' (2) the Chaum Pedersen verifier would accept the triple $(\bar{A}_2, c_2, r_2)$ with ciphertexts C_1 and C_2' and (3) $c = c_1 \oplus c_2$.

Completeness: Without loss of generality, we consider the case where the prover knows first read encryption factor r_1. The prover then runs the Chaum-Pedersen simulator for ciphertexts C_1 and C_2' to obtain the triple $(\bar{A}_2, c_2, r_2)$. The prover then chooses $\bar{A}_2$ as the honest prover would in the Chaum-Pedersen protocol and sends $\bar{A}_1, \bar{A}_2$ to the verifier.

Upon receiving, challenge c from the verifier, the prover chooses c_1 so that $c = c_1 \oplus c_2$ and computes the response r_1 to challenge c_1 as the honest prover would in the Chaum-Pedersen protocol. The verifier will accept because $(\bar{A}_1, c_1, r_1)$ are constructed honestly as in the Chaum-Pedersen protocol and $(\bar{A}_2, c_2, r_2)$ are constructed by the Chaum-Pedersen Simulator. That is, the verifier accepts $(\bar{A}_1, c_1, r_1)$ because the Chaum-Pedersen protocol is complete and the verifier accepts $(\bar{A}_2, c_2, r_2)$ because the Chaum-Pedersen protocol is honest zero-knowledge.

Soundness: Recall that the "special" soundness of the Chaum-Pedersen protocol implies that if for some $\bar{A}$ you have more than one valid c, r pair then you can extract a witness to the fact that the two ciphertexts have the same plaintext (in particular, the re-encryption factor). If in the above protocol the prover can answer some ϵ fraction of possible challenges where ϵ is non-negligible, then one can sample random challenges c and in polynomial time find a pair of challenges $c \neq c'$ such that the prover correctly answers responds to both c and c'. Since $c = c_1 \oplus c_2$ and $c' = c_1' \oplus c_2'$ and $c \neq c'$ then either $c_1 \neq c_1'$ or $c_2 \neq c_2'$. Therefore one can extract either a witness that $C_1 \approx C_1'$ or a witness that $C_1 \approx C_2'$ and hence the disjunction $[(C_1 \approx C_1') \vee (C_1 \approx C_2')]$ must be true.

Honest Zero-Knowledge: The simulator picks c_1 and c_2 independently at random and selects c so that $c = c_1 \oplus c_2$. The simulator then constructs $\bar{A}_1$ and r_1 corresponding to challenge c_1 in the same way that the Chaum-Pedersen simulator would. Similarly, the simulator constructs $\bar{A}_2$ and r_2 corresponding to challenge c_2 in the same way that the Chaum-Pedersen simulator would. Observe that since c_1 and c_2 are chosen independently, c is a random element of Z_q*.

Remark: In the above protocol, one can interpret c_1 and c_2 as a sharing of secret c. This is a special case of a general connection between secret sharing schemes and proofs of monotone Boolean formulas. See [CDS94] for more details.

[3] Here we denote by $\oplus$ the operation of addition in the ring Z_q^*.

Remark: Since the above protocol is honest zero knowledge, it is also witness indistinguishable. Recall the witness indistinguishable proofs can be composed in parallel and remain witness indistinguishable.

2 Going From 2 to n

Here we use sorting networks to build an n by n verifiable mix from many 2 by 2 verifiable mixes.[4] A sorting network is a circuit with n input wires and n output wires consisting of 2 by 2 *comparator* gates. The operation of a comparator is as follows: If the two input wires have values x and y then the comparator outputs $\max(x, y)$ on the "top" output wire and $\min(x, y)$ on the "bottom" output wire.

$$\begin{array}{ccc} x - - - - - & \circ & - - - - - \max(x,y) \\ & | & \\ & | & \\ & | & \\ y - - - - - & \circ & - - - - - \min(x,y) \end{array}$$

Such a network of comparators is a valid sorting network if for any possible set of values on the n input wires, the values on the n output wires are in sorted order. There exist sorting networks with $n \log^2 n$ comparators. More efficient sorting networks also exist, but are very complex.

The property of a sorting network which we make use of is that the sorting network can realize all possible permutations. Our strategy is to replace each comparator in a sorting network with a 2 by 2 verifiable mix to achieve an n by n verifiable mix. That is, (A) if the prover shows that each comparator in the network is performing a valid permutation then the entire network must be performing a valid permutation and (B) since the comparators in the network are capable of realizing any n by n permutation, the fact that a sorting network is used gives the verifier no information about the permutation being implemented.[5]

Remark: Since the proof that the permutation is valid will follow the structure of the sorting network, it is reasonable for the prover to take this into account when performing the mix. One idea is to have the prover perform the mix as follows: First the prover picks a random meta-tag for each input ciphertext. Then the prover runs the sorting network to sort according to the meta-tags. Then at the end, the prover deletes the meta-tags.

Remark: To obtain the challenges for the proof, you could either have everyone commit to challenges ahead of time (i.e, have everyone commit to a share of the master challenge) and

[4] For more information on sorting networks see "Introduction to Algorithms" by Cormen, Leiserson, Rivest and Stien (MIT Press and McGraw-Hill, 2001). The approach here for building a mix network based on a corresponding sorting network is due to Jakobsson and Juels [JJ99].

[5] Depending on the notion of voter privacy that we are trying to achieve, we may not actually need to use a sorting network here. If we were to use any network of comparators that allowed each input value to reach each output position (but not necessarily capable of realizing every permutation) then we would still achieve a weaker yet meaningful notion of voter privacy. One can imagine settings where one would be willing to accept the weaker notion of voter privacy in exchange for gains in prover efficiency.

then have everyone decommit after the first step of the proofs had been given. Alternatively, you could use the Fiat-Shamir paradigm and obtain the challenges by hashing the first prover message in the protocol. Additionally, when running the mix-net, one could think of having each mix server give a proof of correctness before the next mix-server began working, but in practice it is probably better for all the servers to give proofs after all the mixing has been done (as this allows for greater parallelism).

3 Remark on Batch Sizes:

In a large election it is probably infeasible to run a single mix-net consisting of all voters in the election. Instead, it is probably more reasonable to mix the ballots in batches (possibly corresponding to some geographic region such as a precinct or county). The problem with making batches too large is that it increases the time to perform the mix. The problem with making the batches too small is that voters are anonymous only within their batch. To better understand the efficiency of the above scheme, we roughly estimate its performance on a precinct with a thousand voters.

$$\begin{aligned}
\text{Number of Voters}(n) &= 10^3 \\
\text{Number of Comparators}(nlog^2n) &= 10^5 \\
\text{Modular Exponentiations per Comparator} &= 10 \\
\text{Total Modular Exponentiations} &= 10^6 \\
\text{Modular Exponentiations per Second (on PC)} &= 50 - 100 \\
\text{Total Time to Perform Verifiable Mix} &= 3 - 4 \text{ hours}
\end{aligned}$$

It seems fairly reasonable to have a separate PC for each precinct and 3-4 hours is very reasonable amount of time to have to wait for election results. Therefore, one can imagine the above scheme actually being used in practice. However, Andrew Neff in his paper, "Verifiable Mixing (Shuffling) of ElGamal Pairs", provides an even faster verifiable mixing protocol which requires just $8n + 5$ modular exponentiations to prove and $9n + 2$ modular exponentiations to verify. Dan Boneh says that the Neff mixing protocol is efficient enough to mix batches of 100,000 ballots in about 20 hours.

References

[CDS94] Ronald Cramer, Ivan Damgard, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *Proc. CRYPTO '94*, volume LNCS, pages 174–187. Springer, 1994.

[JJ99] M. Jakobsson and A. Juels. Millimix – mixing in small batches. Technical Report 99-33, DIMACS, June 1999.