Berner Fachhochschule - Technik und Informatik

Verifiable Shuffling of Ciphertexts

E-Voting Seminar

Rolf Haenni

October 15th, 2010

Outline

Outline

Homomorphic Encryption

Key Pair	$(x, y) \in \mathcal{X} \times \mathcal{Y}$	
Plaintext:	$m \in \mathcal{M}$	
Randomness:	$r \in \mathcal{R}$	
Ciphertext:	$e \in \mathcal{E}$	
Encryption:	$E_y : \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{E}$	$e = E_y(m, r)$
Decryption:	$D_x : \mathcal{E} \rightarrow \mathcal{M}$	$m = D_x(e)$
Groups:	$(\mathcal{M}, \cdot, 1)$, $(\mathcal{R}, +, 0)$, $(\mathcal{E}, \cdot, 1)$	
Homomorphism:	$E_y(m_1, r_1) \cdot E_y(m_2, r_2) = E_y(m_1 \cdot m_2, r_1 + r_2)$	
	$E_y(m_1, r_1) \,/\, E_y(m_2, r_2) = E_y(m_1 / m_2, r_1 - r_2)$	

Re-Encryption

Re-encryption: $R : \mathcal{E} \times \mathcal{R} \rightarrow \mathcal{E}$, such that $D_x(R(e, r')) = D_x(e)$ for all $e \in \mathcal{E}$, $r' \in \mathcal{R}$

Example 1: $R(e, r') = e \cdot E_y(1, r') = E_y(m, r) \cdot E_y(1, r') = E_y(m, r + r')$

Example 2: $R(e, r') = e \,/\, E_y(1, r') = E_y(m, r) \,/\, E_y(1, r') = E_y(m, r - r')$

Example 3: $R(e, r') = e \cdot E_y(1, r')^z = E_y(m, r) \cdot E_y(1, zr') = E_y(m, r + zr')$

etc.

ElGamal Cryposystem

Setting: Large primes p, q satisfying $p = 2q + 1$, cyclic sub-group $\mathcal{G} \subseteq \mathbb{Z}_p^*$ of order q, generator $g \in \mathcal{G}$

Key Pair $(x, y) \in \mathbb{Z}_q \times \mathcal{G}$ such that $y = g^x$

Plaintext: $m \in \mathcal{G}$

Randomness: $r \in \mathbb{Z}_q$

Ciphertext: $e \in \mathcal{G} \times \mathcal{G}$

Encryption: $E_y : \mathcal{G} \times \mathbb{Z}_q \rightarrow \mathcal{G} \times \mathcal{G}$

$$E_y(m, r) = (g^r, m \cdot y^r) = (a, b)$$

Decryption: $D_x : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$ $\quad D_x(a, b) = b/a^x = m$

Re-encryption: $R : \mathcal{G} \times \mathcal{G} \times \mathbb{Z}_q \rightarrow \mathcal{G} \times \mathcal{G}$

$$R(a, b, r') = (a, b) \cdot (g^{r'}, y^{r'}) = (g^{r+r'}, m \cdot y^{r+r'})$$

Outline

Shuffling Ciphertexts

Input: Ordered list of ciphertexts $\vec{e} = (e_1, \ldots, e_n)$
Randomness vector $\vec{r}\,' = (r'_1, \ldots, r'_n)$
Permutation $\pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$

Output: Shuffled list $\vec{e}\,' = (e'_1, \ldots, e'_n)$ of re-encrypted ciphertexts, where $e'_i = R(e_{\pi(i)}, r'_i)$

$$\vec{e}\,' = shuffle_{\pi}(\vec{e}, \vec{r}\,') = (R(e_{\pi(1)}, r'_1), \ldots, R(e_{\pi(n)}, r'_n))$$

Verifiable shuffling: Given two lists of encryptions $\vec{e}$ and $\vec{e}\,'$, prove existence of a permutation π, s.t. $\vec{e}\,' = shuffle_{\pi}(\vec{e}, \vec{r}\,')$ for some $\vec{r}\,'$, without revealing any information about π or $\vec{r}\,'$

$\Rightarrow$ Honest-verifier zero-knowledge (HVZK) proof

Repeated Shuffling

- The purpose of shuffling is to create an anonymous list of encryptions of the same plaintexts
- Knowing either $\vec{r}\,'$ or π is sufficient to link $\vec{e}\,'$ with $\vec{e}$
- To prevent a single shuffler from violating the anonymity, the list is shuffled multiple times by different shufflers:

$$\begin{aligned}\vec{e}\,' &= \mathit{shuffle}_{\pi_k}(\cdots(\mathit{shuffle}_{\pi_2}(\mathit{shuffle}_{\pi_1}(\vec{e}, \vec{r}_1'), \vec{r}_2')\cdots), \vec{r}_k') \\ &= \mathit{shuffle}_{\pi}(\vec{e}, \vec{r}\,'),\end{aligned}$$

  for $\pi = \pi_k \circ \cdots \circ \pi_1$ and some $\vec{r}\,' = f_\pi(\vec{r}_1', \ldots, \vec{r}_k')$
- A system of repeated shuffling of ciphertexts with independent shufflers (or mixers) is called re-encryption mixnet

Related Problems

- Shuffling and decrypting: $\vec{e} = (e_1, \ldots, e_n)$

$$\vec{m} = shuffle_\pi(\vec{e}) = (D_x(e_{\pi(1)}), \ldots, D_x(e_{\pi(n)}))$$

$\Rightarrow$ Decryption Mixnet

- Shuffling public keys: $\vec{y} = (y_1, \ldots, y_n) = (g^{x_1}, \ldots, g^{x_n})$

$$\begin{aligned}\vec{y}\,' &= shuffle_\pi(\vec{y}, \alpha) = (y^\alpha_{\pi(1)}, \ldots, y^\alpha_{\pi(n)}) \\ &= (\hat{g}^{x_{\pi(1)}}, \ldots, \hat{g}^{x_{\pi(n)}}), \text{ for } \hat{g} = g^\alpha\end{aligned}$$

$\Rightarrow$ C. A. Neff, *A Verifiable Secret Shuffle and its Application to E-Voting*, CCS'01, pages 116–125, 2001

$\Rightarrow$ Selectio Helvetica

Outline

Naïve Proof of Correct Shuffling

- Given two lists of encryptions $\vec{e}$ and $\vec{e}\,'$, prove existence of a permutation π, s.t. $\vec{e}\,' = \mathit{shuffle}_\pi(\vec{e}, \vec{r}\,')$ for some $\vec{r}\,'$

$$ZKP\left[(\pi, \vec{r}\,') : \vec{e}\,' = \mathit{shuffle}_\pi(\vec{e}, \vec{r}\,')\right]$$

$$= ZKP\left[(\vec{r}\,') : \begin{array}{c} (e'_1 = R(e_1, r'_1) \vee \cdots \vee e'_n = R(e_1, r'_n)) \wedge \\ \vdots \qquad\qquad \vdots \\ (e'_1 = R(e_n, r'_1) \vee \cdots \vee e'_n = R(e_n, r'_n)) \end{array}\right]$$

$$= ZKP\left[(\vec{r}\,') : \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} e'_j = R(e_i, r'_j)\right]$$

- Requires n^2 many plaintext equivalence proofs (PEP):

$$e'_j = R(e_i, r'_j) \;\Leftrightarrow\; D_x(e'_j) = D_x(e_i)$$

Plaintext Equivalence Proofs (PEP)

- Two ciphertexts $e_1 = E_y(m_1, r_1)$ and $e_2 = E_y(m_2, r_2)$ have equivalent plaintexts, $m_1 = m_2$, iff $e_1/e_2 = E_y(1, r_1 - r_2)$

  $\Rightarrow$ $r_1 - r_2$ is the re-encryption randomness (sufficient for PEP)

- In ElGamal, proving $(a, b) = E_y(1, r) = (g^r, y^r)$ is equivalent to proving $\log_g a = \log_y b$ (equality of discrete logarithms)

  $\Rightarrow$ $ZKP\,[(r) : a = g^r \wedge b = y^r]$

- Note that $(a, b) = E_y(1, r)$ implies $a{\cdot}b = (g{\cdot}y)^r$, but not vice versa

- If $z \in_R \mathbb{Z}_q$ is a challenge selected after publishing (a, b), then $(a, b) = E_y(1, r)$ implies $a{\cdot}b^z = (g{\cdot}y^z)^r$ and vice versa

  $\Rightarrow$ $ZKP\,[(r) : a{\cdot}b^z = (g{\cdot}y^z)^r]$

  $\Rightarrow$ $ZKP\,[(r) : A = G^r]$, for $A = a{\cdot}b^z$, $G = g{\cdot}y^z$ (Schnorr protocol)

Naïve Proof for ElGamal

- Public values:

$$\vec{e} = ((a_1, b_1), \ldots, (a_n, b_n)), \;\; \vec{e}\,' = ((a'_1, b'_1), \ldots, (a'_n, b'_n))$$

- Private values: $\vec{r}\,' = (r'_1, \ldots, r'_n)$, π
- Interactive protocol for $ZKP\,[(\pi, \vec{r}\,') : \vec{e}\,' = shuffle_\pi(\vec{e}, \vec{r}\,')]$

  1. Verifier sends challenge $z \in_R \mathbb{Z}_q$ to prover
  2. Prover computes $a_i b_i^z$ and $a'_i b_i'^z$ for all $1 \leq i \leq n$
  3. Prover computes $A_{ij} = (a_i b_i^z)/(a'_j b_i'^z)$ for all $1 \leq i, j \leq n$
  4. Prover computes $G = g \cdot y^z$
  5. Prover generates interactive $ZKP\left[(\vec{r}\,') : \wedge_i \vee_j A_{ij} = G^{r'_j}\right]$

  Verifier accepts proof if A_{ij} and G are correct and if ZKP holds

Schnorr Protocol

- Public values: G, $A = G^r$
- Private value: r
- Interactive Σ-Protocol for $ZKP\,[(r) : A = G^r]$
  1. Prover selects $q \in_R \mathbb{Z}_q$ and sends commitment $t = G^q$ to the verifier
  2. Verifier sends challenge $c \in_R \mathbb{Z}_q$ to the prover
  3. Prover sends response $s = q + cr$ to the verifier

  The verifier accepts conversation (t, c, s), if $G^s = t{\cdot}A^c$
- Testing $G^s = t{\cdot}A^c$ is equivalent to testing $G^s{\cdot}A^{-c} = t$
- We may use multiple exponentiation algorithms for computing $G^s{\cdot}A^{-c}$ more efficiently

Schnorr Protocol: AND-Combination

- Public values: G_1, G_2, $A_1 = G_1^{r_1}$, $A_2 = G_2^{r_2}$
- Private values: r_1, r_2
- Interactive Σ-Protocol: $ZKP\left[(r_1, r_2) : A_1 = G_1^{r_1} \wedge A_2 = G_2^{r_2}\right]$
  1. Prover selects $q_1, q_2 \in_R \mathbb{Z}_q$ and sends $t = (t_1, t_2) = (G_1^{q_1}, G_2^{q_2})$ to the verifier
  2. Verifier sends $c \in_R \mathbb{Z}_q$ to the prover
  3. Prover sends $s = (s_1, s_2) = (q_1 + cr_1, q_2 + cr_2)$ to the verifier

  The verifier accepts conversation (t, c, s), if $G_1^{s_1} = t_1 \cdot A_1^c$ and $G_2^{s_2} = t_2 \cdot A_2^c$
- We may use fixed-exponent exponentiation algorithms for computing A_1^c and A_2^c more efficiently

Schnorr Protocol: OR-Combination

- Public values: G_1, G_2, $A_1 = G_1^{r_1}$, $A_2 = G_2^{r_2}$
- Private value: r_1 (but not r_2)
- Interactive Σ-Protocol: $ZKP\,[(r_1, r_2) : A_1 = G_1^{r_1} \vee A_2 = G_2^{r_2}]$
    1. Prover selects $q_1, q_2 \in_R \mathbb{Z}_q$ and sends $t = (t_1, t_2) = (G_1^{q_1}, G_2^{q_2})$ to the verifier
    2. Verifier sends $c \in_R \mathbb{Z}_q$ to the prover
    3. Prover generates valid conversation (t_2, c_2, r_2)
    4. Prover computes $c_1 = c - c_2$ and $s_1 = q_1 + c_1 r_1$
    5. Prover sends $c = (c_1, c_2)$ and $s = (s_1, s_2)$ to the verifier

The verifier accepts conversation (t, c, s), if $G_1^{s_1} = t_1 \cdot A_1^{c_1}$, $G_2^{s_2} = t_2 \cdot A_2^{c_2}$, and $c = c_1 + c_2$

Outline

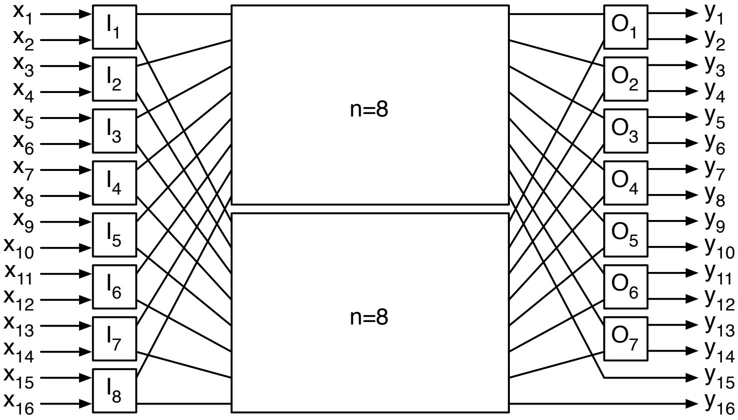
Millimix

- Millimix is a method for proving the correctness of a shuffle in $O(n \log n)$ space/time
- M. Jakobsson, A. Juels, *Millimix: Mixing in Small Batches*, Technical Report 99-33, DIMACS, 1999
- The idea is to decompose the n-permutation π into a network of $O(n \log n)$ many 2-permutations π_i
- There are two possibilities $\bar{\pi} = (1, 2)$ or $\tilde{\pi} = (2, 1)$
- Since there are $n!$ many different n-permutations, we need at least $\log_2 n!$ many 2-permutations
- Proving correct shuffling for all involved π_i will prove correct shuffling of π

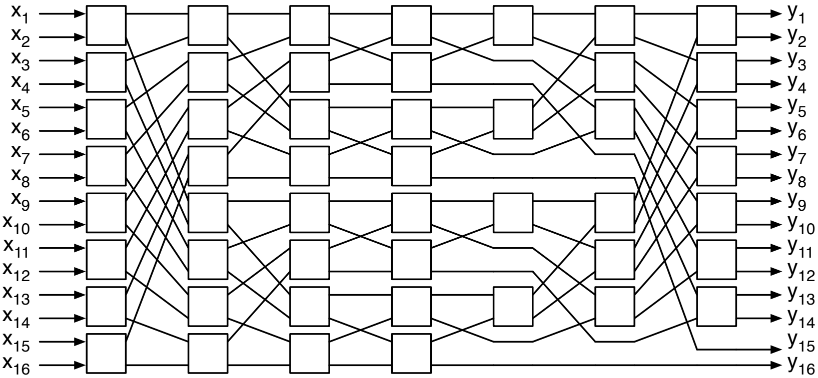
Network Construction

- A. Waksman, *A Permutation Network*, Journal of the ACM, 15(1), pages 159–163, 1968
- The idea is to recursively decompose the n-permutation into two $\frac{n}{2}$-permutations
- The two $\frac{n}{2}$-permutations are connected by $\frac{n}{2}$ input nodes and $\frac{n}{2}-1$ output nodes (total $n-1$ nodes)
- This produces $F(n) = (n-1) + 2(\frac{n}{2}-1) + 4(\frac{n}{4}-1) + \cdots$ many nodes
- For $n = 2^k$, this implies $F(n) = n{\cdot}(\log_2 n - 1) + 1 \in O(n \log n)$
- Note that $\log_2 n! \leq F(n) < \log_2 (n+1)!$ for $n \geq 2$

Network Construction: $n = 16$



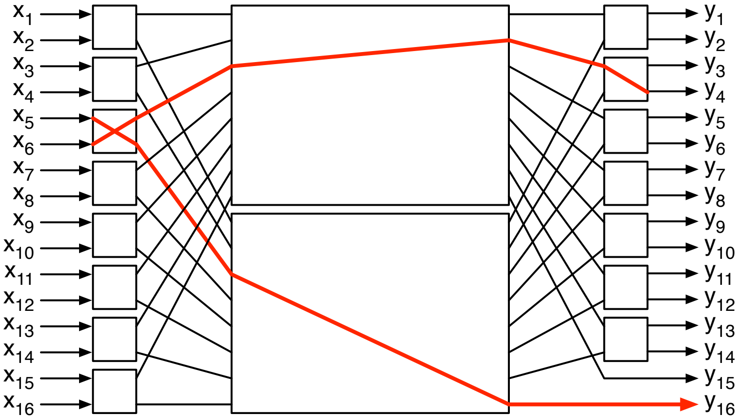
Network Construction: $n = 16$



Total nodes: $F(n) = 16 \cdot (\log_2 16 - 1) + 1 = 49$

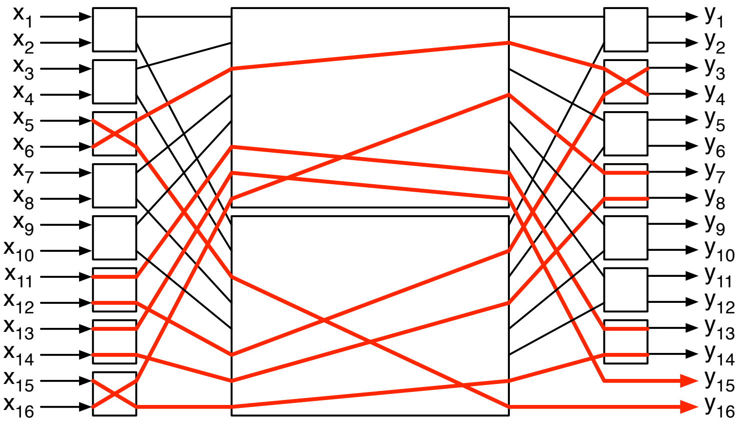
Network Construction: $n = 16$

Example: $\pi = (3, 9, 12, 6, 7, 1, 16, 14, 8, 2, 10, 4, 11, 15, 13, 5)$



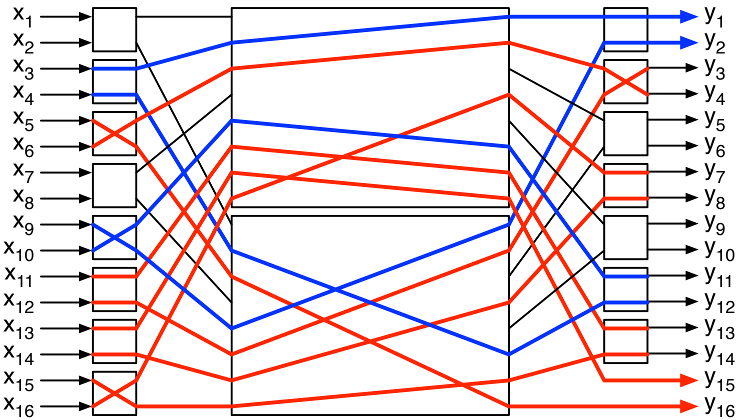
Network Construction: $n = 16$

Example: $\pi = (3, 9, 12, 6, 7, 1, 16, 14, 8, 2, 10, 4, 11, 15, 13, 5)$



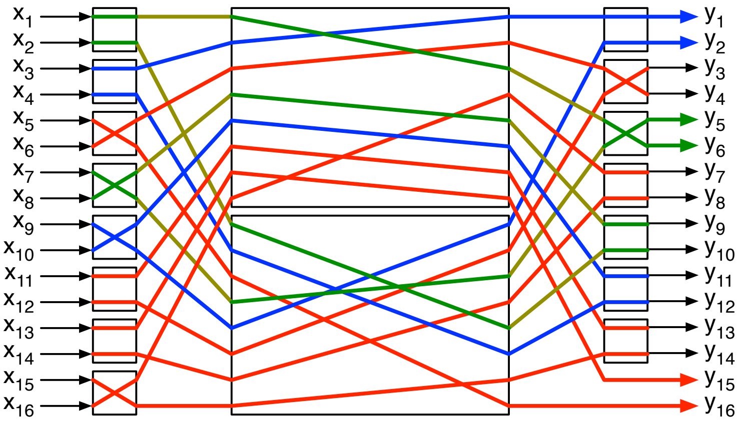
Network Construction: $n = 16$

Example: $\pi = (3, 9, 12, 6, 7, 1, 16, 14, 8, 2, 10, 4, 11, 15, 13, 5)$



Network Construction: $n = 16$

Example: $\pi = (3, 9, 12, 6, 7, 1, 16, 14, 8, 2, 10, 4, 11, 15, 13, 5)$



Proving Correctness of 2-Shuffe

- Given two lists of encryptions $\vec{e} = (e_1, e_2)$ and $\vec{e}\,' = (e'_1, e'_2)$, prove that $\vec{e}\,' = \mathit{shuffle}_{\bar{\pi}}(\vec{e}, \vec{r}\,')$ or $\vec{e}\,' = \mathit{shuffle}_{\tilde{\pi}}(\vec{e}, \vec{r}\,')$ for some $\vec{r}\,' = (r'_1, r'_2)$

$$ZKP\left[(\vec{r}\,') : \vec{e}\,' = \mathit{shuffle}_{\bar{\pi}}(\vec{e}, \vec{r}\,') \vee \vec{e}\,' = \mathit{shuffle}_{\tilde{\pi}}(\vec{e}, \vec{r}\,')\right]$$

$$= ZKP\left[(r'_1, r'_2) : \begin{array}{c} (e'_1 = R(e_1, r'_1) \vee e'_2 = R(e_1, r'_2)) \\ \wedge \\ (e'_1 = R(e_2, r'_1) \vee e'_2 = R(e_2, r'_2)) \end{array}\right]$$

$$= ZKP\left[(r'_1, r'_2) : \begin{array}{c} (e'_1 = R(e_1, r'_1) \vee e'_2 = R(e_1, r'_2)) \\ \wedge \\ e'_1 \cdot e'_2 = R(e_1 \cdot e_2, r'_1 + r'_2) \end{array}\right]$$

Outline

Conclusion and Outlook

- Millimix runs in $O(n \log n)$ time and space (per mixer)
- The methods of Furukawa/Sako, Neff, Groth, and Wikström run in $O(n)$ time, but with higher constants
- Millimix might still be a good choice for small n ("small batches")
- All competing approaches are far more complicated
  - → Furukawa/Sako, Wikström: permutation matrices
  - → Neff, Groth: homomorphic commitments, root-permutable polynomials
- Open question: what is the best method for shuffling public keys in Selectio Helvetica?