An Implementation of a Mix-Net Based Network Voting Scheme and Its Use in a Private Organization

Jun Furukawa, Kengo Mori, and Kazue Sako

NEC Corporation
j-furukawa@ay.jp.nec.com, ke-mori@bx.jp.nec.com, k-sako@ab.jp.nec.com

Abstract. We discuss an implementation of a network voting scheme based on mix-net technology. We employed the scheme presented at Financial Cryptography 2002, but replaced the numeric computations with those on a elliptic curve. As a result, we obtained three times speed up and data length shortening to one third. The system has been employed in a private organization with roughly 20,000 voters since 2004.

1 Introduction

The three basic requirements on a secure network voting system are detection of faulty voters, detection of faulty centers, and vote secrecy. Among other voting protocols achieving these three requirements, such as blind signature based schemes [FOO92, Sak94, OMAFO] and homomorphic encryption based schemes [CY85, SK94, CFSY96, CGS97, DJ01], we have chosen the mix-net based scheme [PIK93, SK95, Abe99, FS01, Ne01] for implementing our voting system.

Although the scheme requires rather large amount of computation on mixers who shuffle and decrypt the encrypted votes, and who we need to assume not to collude with all other mixers, the scheme offers many desirable features for voters and for administrators who manage voting system:

- it enjoys flexibility in representing a vote, unlike homomorphic encryption based scheme where the design of the system depends heavily on the number of choices in each vote.
- the voters can simply vote-and-go and require only a small computational ability.
- by having authorities prove the correctness of their procedures, it achieves public verifiability, that is, anyone can verify the correctness of the tally.

The cost that is paid for these properties is the computational cost to generate proofs assuring correctness of shuffling and decryption. However, it can be relaxed by elaborating computational algorithms such as use of "fixed-base comb method" and "Simultaneous multiple exponentiation method" exposed in [MOV]

As a result, the system was able to produce results that were verified correct within *6.6* minutes following a vote participated in by ten thousand voters, with three mixers each on a 1GHz PC.

D. Chaum et al. (Eds.): Towards Trustworthy Elections, LNCS 6000, pp. 141–154, 2010.

For the proof, we used the scheme proposed in [FMMOS02]. Although the scheme can not prove to have zero-knowledge property, it has complete permutation-hiding property as discussed in [F04].

We note that the property of receipt-freeness is not achieved in our system. Also, the privacy of abstaining is not currently being supported. On the other hand, the administrators knowing who have voted and who have not, help to send reminders to those who have not voted.

2 The Mix-Net Based Voting Scheme

2.1 Overview

The mix-net based voting schemes achieve anonymity of votes by distributing the decryption key among multiple authorities called mixers. Voters send signed but encrypted votes. These votes will be processed by the mixers, who shuffles the votes and decrypt them. After all the mixers have done their work, the encrypted vote will be completely decrypted, but its original owner can not be identified due to the shuffles performed at every mixer. By providing shuffle-and-decrypt proofs, anyone can verify the output is the correct decryptions of the shuffled valid votes. The proofs should be made in a way it will not reveal the permutation used in the shuffle nor the decryption key, so it would not infringe the vote anonymity.

2.2 Model

We involve five kinds of players, which are

1. Election policy committee
2. Voting center
3. Shuffling management center
4. Shuffling center(mixer)
5. Voters

The election policy committee will be responsible for any fraud caused by the voting center, the shuffling management center, and the shuffling centers. The election policy committee does not engage in an actual run of the electronic voting. It takes part in determining election and security policies, and assignment of the centers. The committee authorizes the output computed by the other centers, such as the parameters determined in a set-up phase and the final tally.

The voting center is in charge of handling transactions with the voters. It will announce the voting procedures, collects pro forma votes from the authorized voters, issues receipts of the collected votes, and announces the result of the tally. The voting center will receive the result of the tally by sending the list of collected votes to the shuffling management center.

The shuffling management center is responsible for decrypting and tallying the list sent from the voting center, in collaboration with the shuffling centers(mixers). The shuffling management center passes the list to the first shuffling center, and collects his answer which will be sent to of the next shuffling

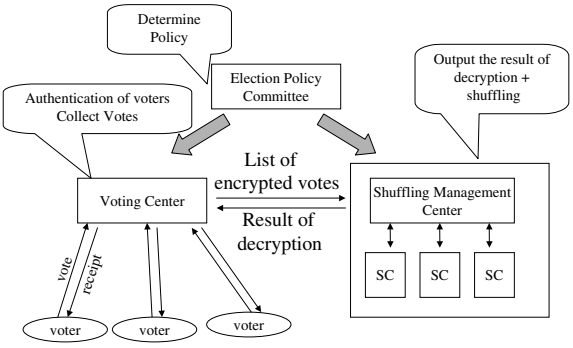

Fig. 1. System Configuration of Mix-net

center, and repeats the process until all the assigned shuffling centers shuffle and decrypt the list. The shuffled result of decryption will be sent back to the voting center. The shuffling management center is also responsible for composing a public key in the set-up phase, again in collaboration with the shuffling centers.

The shuffling center, whose other name is the mixer, is responsible for secure management of the secret key generated in the set-up phase, and conducting decryption using the key. He is also responsible for randomly shuffling the list and keeping the permutation used in a shuffle confidential.

We require for the universal verifiability, that any party can verify that all centers conducted correctly based on the policy approved by the election policy committee. Our goal in vote privacy is that it will not be infringed as long as at least one shuffling center remains honest.

Figure 1 illustrates how these players constitute an voting system. We note that the roles of the voting center and the shuffling management center can be played by one entity.

2.3 Protocol

In this subsection we describe the procedure to set-up, to encrypt votes, and to tally the votes.

In the sequel, we assume there are m shuffling centers and n voters. All the communication between the centers are digitally signed based on a public key infrastructure.

Set-Up

1. The election policy committee will determine the parameters $(q, \mathbf{E}, g)$ which will be used in ElGamal cryptosystem on an elliptic curve $\mathbf{E}$. The numbers q is a prime order of the elliptic curve $\mathbf{E}$ and g is a generator.
   The shuffling management center announces the authorized parameters $(q, \mathbf{E}, g)$ to all the shuffling centers. The j-th shuffling center, SC_j, will randomly choose x_j mod q as his secret key, and report his public key $y_j = [x_j]g$

to the shuffling management center. The report is accompanied by the proof y'_j, r_j which ensures that SC_j indeed knows the secret x_j corresponding to y_j. The proof will be generated by SC_j as follows.

$$\begin{aligned} y'_j &= [\beta_j]g \\ c_j &= \mathcal{H}(p, q, g, y_j, y'_j) \\ r_j &= c_j x_j + \beta_j \bmod q \end{aligned}$$

with a randomly generated $\beta_j \in \mathbf{Z}/q\mathbf{Z}$.

2. The shuffling management center will verify the proof y'_j, r_j for each public key $y_j (j = 1, \cdots, m)$ as follows.

$$\begin{aligned} c_j &= \mathcal{H}(p, q, g, y_j, y'_j) \\ [r_j]g - [c_j]y_j &= y'_j \\ y_j \in \mathbf{E} \ &, \ y_j \neq \mathcal{O} \end{aligned}$$

The verified public keys are combined to compose the common public key Y.

$$Y = \sum_{j=1}^{m} y_j$$

The proof for each public key is necessary to ensure that the common public key Y corresponds to each of the secret keys that the mixers are aware of, not those generated under a control of an adversary.

3. The election policy committee will certify the public keys y_j and Y properly generated as above.

Encryption of Votes

The $Voter_i$ will use the parameters Y and $(q, \mathbf{E}, g)$ certified by the election policy committee and encrypt his vote m_i as follows. (We assume here that m_i is in $\mathbf{E}$.)

$$(G_i, M_i) = ([\bar{r}_i]g, m_i + [\bar{r}_i]Y)$$

where $\bar{r}_i$ is an element randomly chosen by the $Voter_i$, and ID_i is information that identifies the voter. He may then prove the knowledge of m_i by generating the proof α_i, t_i by

$$\begin{aligned} \alpha_i &= [\gamma_i]g \\ c_i &= \mathcal{H}(p, q, g, Y, G_i, \alpha_i, ID_i) \\ t_i &= c_i \bar{r}_i + \gamma_i \bmod q \end{aligned}$$

with a randomly generated γ_i. This proof ensures that the plaintext awareness property: that is, a voter who knows the content of his vote has generated the encrypted vote. A vote duplication attack by copying someone else's encrypted vote will be thwarted here.

The voting center will verify that the voter is eligible to vote. It will verify that the proof satisfies

$$c_i = \mathcal{H}(p, q, g, Y, G_i, \alpha_i, ID_i)$$
$$[t_i]g - [c_i]G_i = \alpha_i$$

and that the elements G_i and M_i are both in $\mathbf{E}$. If everything is verified, then it can optionally send back a receipt of acceptance. Such a receipt cuts in two ways: it will add confidence to the voter that the center indeed accepted his vote and will be an evidence for any disputes on vote delivery. On the other hand, it will serve as a receipt in vote-buying or coercing scenario.

Tallying

The voting center will send the list of accepted votes from each voters $(G_i, M_i)_{i=1,\cdots,n}$ to the shuffling management center. The shuffling management center will verify that all of each component is in $\mathbf{E}$, and rename them to be $(G_i, M_i) = (G_i^{(1)}, M_i^{(1)})$ for all i, which will be the input to the first shuffling center $SC1$.

The list $(G_i^{(j)}, M_i^{(j)})_i$ will be sent to SC_j. His response will be verified by the shuffling management center and will be renamed to $(G_i^{(j+1)}, M_i^{(j+1)})_i$ and sent to the next shuffling center. The response from the last shuffling center, SC_m will be verified and sent back to the voting center.

Below, we describe the procedures of each shuffling center.

1. SC_j will receive the list $(G_i^{(j)}, M_i^{(j)})_i$. He will choose a random permutation $\pi^{(j)}$ and permute the input list $(G_i^{(j)}, M_i^{(j)})_i$ and achieve the list $(\bar{G}_i^{(j)}, \bar{M}_i^{(j)})_i$ as follows:
$$(\bar{G}_i^{(j)}, \bar{M}_i^{(j)})_i = (G_{\pi^{(j)}(i)}^{(j)}, M_{\pi^{(j)}(i)}^{(j)})_i$$
2. The above permutation only changes the order of the ciphertexts, so it is easy to trace the permutation. In order to hide the permutation, we need to change the *look* of the ciphertext. The following procedure changes the look without changing the message hidden in the ciphertext.
First, SC_j combines the public keys of the subsequent shuffling centers as
$$Y_j = \sum_{\ell=j}^{m} y_\ell.$$
For each of $(\bar{G}_i^{(j)}, \bar{M}_i^{(j)})$, he chooses a random element $s_i^{(j)} \bmod q$ and obtains $({G'}_i^{(j)}, {M'}_i^{(j)})_i$ by
$${G'}_i^{(j)} = \bar{G}_i^{(j)} + [s_i^{(j)}]g$$
$${M'}_i^{(j)} = \bar{M}_i^{(j)} + [s_i^{(j)}]Y_j$$

3. SC_j will decrypt each of $(G'^{(j)}_i, M'^{(j)}_i)$ using his secret key x_j as follows:

$$M''^{(j)}_i = M'^{(j)}_i - [x_j]G'^{(j)}_i \quad G''^{(j)}_i = G'^{(j)}_i$$

The list $(G''^{(j)}_i, M''^{(j)}_i)_i$ will be returned to the shuffling management center.

Proving Correctness

Details of procedure for mixers to prove they have correctly shuffled and decrypted the input is described in the next section.

3 Details of Correctness Proof

For simplicity, we concentrate on one shuffling center and denote his secret key as x. We represent by $\bar{y}$ the product of the public keys of subsequent centers. What we need to prove is the correctness of the following shuffle-and-decrypt procedure.

Given n ciphertexts $(G_i, M_i)_i$, where all $\{G_i\}$ and $\{M_i\}$ are in $\mathbf{E}$, the shuffling center randomly chooses a permutation π and a random element $s_i \in_U \mathbf{Z}/q\mathbf{Z}$ to obtain shuffle-and-decrypt result as follows:

$$(G'_i, M'_i) = \left([s_i]g + G_{\pi(i)}, [s_i]\bar{y} + M_{\pi(i)} - [x]G'_i\right)$$

for $i = 1, \ldots, n$.

3.1 Generation of the Proof

We now provide the scheme to generate a proof that the shuffling center (which will be denoted as the prover in the sequel) indeed shuffled and decrypted honestly.

We describe the scheme in a non-interactive way, where a challenge from a verifier is given as an output of some universal one-way hash functions. We assume here that all elements of input ciphertexts (G_i, M_i) and output ciphertexts (G'_i, M'_i) are in $\mathbf{E}$.

To prove (G'_i, M'_i) are generated correctly from (G_i, M_i), the prover computes the following equations for randomly chosen $z, z_i, \rho, \sigma, \tau, \lambda$ and $\lambda_i, z' \in_U \mathbf{Z}/q\mathbf{Z}$ $(i = 1, \ldots, n)$: We use $\mathcal{H}$ and $\tilde{\mathcal{H}}$ to denote universal one-way hash functions which output an element of $\mathbf{Z}/q\mathbf{Z}$ and $\mathbf{E}$, respectively.

$$\tilde{g} = \tilde{\mathcal{H}}(p, q, g, Y, 0), \quad \tilde{g}_i = \tilde{\mathcal{H}}(p, q, g, Y, i)$$
$$v = [\rho]g, \quad w = [\sigma]g, \quad t = [\tau]g, \quad u = [\lambda]g, \quad u_i = [\lambda_i]g$$
$$\tilde{g}'_i = [s_i]\tilde{g} + \tilde{g}_{\pi(i)}, \qquad \tilde{g}' = [z]\tilde{g} + \sum_{j=1}^{n}[z_j]\tilde{g}_j$$
$$g' = [z]g + \sum_{j=1}^{n}[z_j]G_j, \qquad m' = [z]\bar{y} + \sum_{j=1}^{n}[z_j]M_j$$

$$\dot{t}_i = [3z_{\pi(i)} + \tau\lambda_i]g, \qquad \dot{v}_i = [3{z_{\pi(i)}}^2 + \rho s_i]g$$
$$\dot{v} = [\sum_{j=1}^{n} {z_j}^3 + \tau\lambda + \rho z]g$$
$$\dot{w}_i = [2z_{\pi(i)} + \sigma s_i]g, \qquad \dot{w} = [\sum_{j=1}^{n} {z_j}^2 + \sigma z]g$$
$$\begin{aligned} c_i = \mathcal{H}(p, q, g, \bar{y}, \tilde{g}, \{\tilde{g}_j\}, (G_j, M_j)_j, (G'_j, M'_j)_j, \\ \tilde{g}', (\tilde{g}'_j)_j, g', m', v, w, t, u, (u_j)_j, \\ (\dot{t}_j)_j, \dot{v}, (\dot{v}_j)_j, \dot{w}, (\dot{w}_j)_j, i) \end{aligned} \tag{1}$$
$$r_i = c_{\pi^{-1}(i)} + z_i, \qquad r = \sum_{j=1}^{n} s_j c_j + z \bmod q$$
$$\lambda' = \sum_{j=1}^{n} \lambda_j {c_j}^2 + \lambda \bmod q$$
$$\zeta = \sum_{j=1}^{n} [c_j] G'_j, \qquad \eta = [x]\zeta \tag{2}$$
$$y' = [z']g, \qquad \eta' = [z']\zeta \tag{3}$$
$$c' = \mathcal{H}(p, q, g, y, \zeta, \eta, y', \eta') \tag{4}$$
$$r' = c'x + z' \bmod q \tag{5}$$

The prover send the proof $g', m', \tilde{g}', \tilde{g}'_i, v, w, t, u, u_i, \dot{t}_i, \dot{v}_i, \dot{v}, \dot{w}_i, \dot{w}, r, r_i,\ \lambda', \eta, \eta', y', r'$ $(i = 1, \ldots, n)$ to the verifier along with $(G'_i, M'_i)_i$.

3.2 Verifications of the Proof

The verifier first computes $(c_i)_{i=1,\ldots,n}$ according to Eq.(1). Next, the verifier compute

$$\zeta = \sum_{j=1}^{n} [c_j] G'_j$$

and generate c' according to Eq.(4). The verifier accepts the proof if all of the following equations hold.

$$v, t, w \in \mathbf{E}$$
$$[r]g + \sum_{j=1}^{n} [r_j] G_j = g' + \zeta$$
$$[r]\bar{y} + \sum_{j=1}^{n} [r_j] M_j = \eta + m' + \sum_{j=1}^{n} [c_j] M'_j$$
$$[r]\tilde{g} + \sum_{j=1}^{n} [r_j] \tilde{g}_j = \tilde{g}' + \sum_{j=1}^{n} [c_j] \tilde{g}'_j$$

$$[\lambda']g = u + \sum_{j=1}^{n}[{c_j}^2]u_j$$

$$[\lambda']t + [r]v + [\sum_{j=1}^{n}({r_j}^3 - {c_j}^3)]g = \dot{v} + \sum_{j=1}^{n}[{c_j}^2]\dot{t}_j + \sum_{j=1}^{n}[c_j]\dot{v}_j$$

$$[r]w + [sumj({r_j}^2 - {c_j}^2)]g = \dot{w} + \sum_{j=1}^{n}[c_j]\dot{w}_j$$

$$[r']g = [c']y + y' \ , \ [r']\zeta = [c']\eta + \eta'.$$

3.3 Complete Permutation Hiding

We discuss here the notion of *complete permutation hiding* (CPH) as a core requirement of unlinkability in verifiable shuffle-decryption. If a verifiable shuffle-decryption is CPH, honest verifiers will learn nothing new about its permutation from an interaction with a prover in an **overwhelming** number of cases of random tape that a prover has chosen uniformly and randomly, whereas, if the protocol is zero-knowledge, verifiers will learn nothing new in **every** case of the random tape. In other words, we define CPH so that verifiers learn nothing about the permutation in an overwhelming number of cases of common input X_n and witness W_n that the generator G_R (defined below) outputs.

Let I_n be a *set* of domain parameters $1^n, q, \mathbf{E}$, where q is prime and is of the length of the polynomial of n, and $\mathbf{E}$ is an elliptic curve of an order q, private key $\bar{x}$, plain texts $\{M_i \in \mathbf{E}\}_{i=1,\ldots,k}$, and random tape Z_n. Let $enc(U)$ be an *encoding of a probabilistic polynomial time (PPT) Turing machine* U which generates cipher-texts $(g_i, m_i)_{i=1,\ldots,k}$ input to the shuffle-decryption procedure. We assume the existence of a knowledge extractor that can concurrently extract $\{\bar{r}_i\}_{i=1,\ldots,k}$ such that $[\bar{r}_i]g_0 = g_i$ from U. This assumption is satisfied if all generators of cipher-texts are imposed to run a concurrent proof of knowledge of $\bar{r}_i$, and such a compulsion prevents an adaptively chosen cipher-text attack.

Definition 1. *Given* $I_n(= \{1^n, q, \mathbf{E}, \bar{x} \in \mathbf{Z}/q\mathbf{Z}, \{M_i \in \mathbf{E}\}_{(i=1,\ldots,n)}, Z_n\})$ *and* $enc(U)$, instance Generator G_R *chooses* $g_0 \in_R \mathbf{E}, x' \in_R \mathbf{Z}/q\mathbf{Z}$, $\{s_i \in_U \mathbf{Z}/q\mathbf{Z}\}_{i=1,\ldots,k}$, *and a permutation* π *uniformly and randomly and computes;*

$$m_0 = [x' + \bar{x}]g_0, y = [x']g_0$$
$$(g_i, m_i) = U(I_n, g_0, y) \in \mathbf{E} \times \mathbf{E}$$
$$(g'_i, m'_i) = ([s_i]g_0 + g_{\pi^{-1}(i)}, [-x']g_i + [s_i]m_0 + m_{\pi^{-1}(i)}).$$

G_R *then outputs common input* X_n *and witness* W_n*:*

$$X_n = \{q, \mathbf{E}, y, \bar{x}, g_0, m_0, \{(g_i, m_i)\}_{(i=1,\ldots,n)}, \{(g'_i, m'_i)\}_{(i=1,\ldots,n)}\},$$
$$W_n = \{\pi, \{s_i\}_{(i=1,\ldots,n)}, x'\}.$$

In the above definition, U is a PPT Turing machine that plays the role of (malicious and colluding) players who generate cipher-texts $\{(g_i, m_i)\}$. Although U is

determined before the public parameter is generated, it does not lose generality because it has this public parameter as an input. In a case where U realizes honest players, it outputs

$$(g_i, m_i) = ([\bar{r}_i]g_0, M_i + [\bar{r}_i]m_0)$$

using random numbers $\{\bar{r}_i\}_{i=1,\ldots,k}$ generated from the random tape Z_n.

We say X_n and W_n *satisfy relation* R if the following equations are satisfied:

$$m_0 = [x' + \bar{x}]g_0, y = [x']g_0$$
$$(g'_i, m'_i) = ([s_i]g_0 + g_{\pi^{-1}(i)}, [-x']g_i + [s_i]m_0 + m_{\pi^{-1}(i)}).$$

We denote this fact as $(X_n, W_n) \in R$. If there exists a witness W_n for a common input X_n that satisfies $(X_n, W_n) \in R$, common input X_n is a *correct shuffle-decryption*. Generator G_R outputs such a X_n.

Definition 2. *Let* $View_V^P(X_n, W_n)$ *be* V*'s* view of an interaction *with* P*, which is composed of the common input* X_n*, messages* V *receives from* P*, random tape input to* V*, and messages* V *sends to* P *during joint computation employing* X_n*, where* P *has auxiliary input* W_n *s.t.,* $(X_n, W_n) \in R$*.* $View_V^P$ *is an abbreviation of* $View_V^P(X_n, W_n)$*.*

We consider the case when a semi-honest verifier may collude with malicious players who encrypt the ciphertexts and other provers who shuffle and decrypt in the same mix-net. Such a verifier and players may obtain partial information regarding the plain texts $\{M_i\}$, private key $\bar{x}$ (the sum of other prover's private keys in the mix-net), random tapes of players, and even a part of the permutation π in addition to $View_V^P$. Moreover, they may obtain the results of other shuffle-decryptions executed by the same prover.

Then it is reasonable to describe this extra information as $H(I_n, enc(U), X_n, \pi)$ and input cipher-texts generated by the malicious player as $U(I_n, g_0, y)$ using PPT Turing machines $H(\cdot)$ and $U(\cdot)$. Note that $\{s_i\}$ are not included in the arguments of H, because we consider only the case where the prover never reveals these values to any one and the case where the prover never uses the same $\{s_i\}$ for other shuffle-decryptions.

Even though the verifier and the players may obtain the results of other shuffle-decryptions executed by the same prover who uses x', we do not include x' into the input of U and H. Instead, we assume that there exists a PPT Turing machine K such that the distribution of $View_V^P$ for such H and U and that of $K(I_n, g_0, y, enc(U), \pi)$ are the same. We denote this as $View_V^P \approx K(I_n, g_0, y, enc(U), \pi)$. The exclusion of x' is crucial because it enables us to consider the security of shuffle-decryption over the distribution of X_n i.e., of x'.

We describe information about the permutation π that verifiers try to learn as $f(\pi)$ using PPT Turing machine f. This description can be justified because the expression $f(\pi)$ is sufficient to express any bit of π and any kind of check sum for π.

Now we can say that a verifiable shuffle-decryption protocol hides its permutations completely with respect to G_R - i.e., CPH occurs - if there exists a probabilistic polynomial time algorithm E'^E (which has black box access to E) with inputs X_n and $H(I_n, enc(U), X_n, \pi)$ that suffers no disadvantage with respect to learning anything about the permutations compared to any probabilistic polynomial time verifier E having input $View_V^P$ and $H(I_n, enc(U), X_n, \pi)$. This leads to,

Definition 3. (complete permutation hiding) *A verifiable shuffle decryption protocol* (P, V, G_R) *achieves* complete permutation hiding *if*

$$\begin{aligned}
&\exists E'^E \forall E \forall H \forall f \forall U \forall c > 0 \exists N \forall n > N \forall I_n \\
&\Pr[E(View_V^P, H(I_n, enc(U), X_n, \pi)) = f(\pi)] \\
&< \Pr[E'^E(X_n, H(I_n, enc(U), X_n, \pi)) = f(\pi)] + \frac{1}{n^c}, \qquad (6) \\
&\textit{and} \\
&\exists K \quad View_V^P \approx K(I_n, g_0, y, enc(U), \pi)
\end{aligned}$$

where E', E, H, f, U, K *are PPT Turing machine. The left probability in Eq.(6) is taken over the distribution of the random tapes input to* G_R,[1] P, V, H, *and* E. *The right probability in Eq.(6) is taken over the distribution of the random tapes input to* G_R, H, E', *and* E. E' *may use* E *as a black box.*

If the verifiable shuffle-decryption protocol is CPH, we can say that for **every** input ciphertexts set $\{(g_i, m_i)\}$ and its corresponding output ciphertexts set $\{(g'_i, m'_i)\}$, whatever an honest verifier who has partial information $(H(I_n, enc(U), X_n, \pi))$ about the common input (X_n), can learn about the permutation (π) after interacting with a prover, can also - in an **overwhelming** number of cases of common input (X_n)- be efficiently computed from that common input (X_n) and that partial information $(H(I_n, enc(U), X_n, \pi))$ alone using a PPT Turing machine E' without interaction with the prover as long as the prover has chosen the private key x', permutation π, and random numbers $\{s_i\}$ uniformly and randomly.

Note that we are considering the case even where malicious and colluding players, who have the results of other shuffle-decryptions with the same x', are engaged in generating $\{(g_i, m_i)\}$ of common input. Hence, CPH guarantees security when shuffle-decryptions with the same private key are repeatedly executed[2].

[1] Since the probability is taken over a distribution containing x', we have excluded any adversary who knows x'.

[2] The definition of shuffle-decryption stated in [FMMOS02] is "No polynomially bounded adversary can compute any partial information of the permutation from the protocol". Unlike our new definition, this definition does not mention the case where the verifier has already obtained partial information before the protocol begins and where the shuffle-decryptions with the same private key are repeatedly executed. These cases seem to occur quite often.

Table 1. Processing time and proof size

	Total Time		Length of Proof	
number of voters	10,000	100,000	10,000	100,000
EC Implementation	6.6 min	1 hr 7 min	4.8Mbyte	48Mbyte
Mod p Implementation([FMMOS02])	20 min	3 hrs 44 min	12.6Mbyte	126Mbyte
Tally without proof(EC)	69 sec	12 min	–	–
Tally without proof(Mod p)	8 min	1 hrs 10 min	–	–

4 Result of Implementation

We have evaluated the system under the following conditions:

- Key Size $|q| = 160$.
- Security parameter k=160
- The number of shuffling centers $m = 3$.
- CPU:PentiumIII 1GHz, memory 256Mbyte for each of mixers and shuffling management center.
- Communication Line 100baseTX

As a result, with a ten thousand voters the system can output a certified tally in 6.6 minutes and with a hundred thousand voters within 67 minutes, including data transmission time. Less than one-fifth of the time are required for computing the tally, and the rest of the time is devoted to proving and verifying the proof.

Table 1 compares the implementation results on Elliptic Curve and that on modular p arithmetic reported in [FMMOS02]. Modular arithmetic requires the length of p to be 512 where as EC implementation requires 160. This affects the speed up in tallying and the shortening the proof size, both in the factor of 3.

We describe how we measured the tally. Since proving is the one that takes the most of the time, we introduced parallel scheduling. That is, a shuffling center returns the result of his shuffle-and-decrypt procedure to the shuffling management center before he starts proving the correctness of his result. The shuffling management center verifies the signature on the result and forwards the result to the next shuffling center. Thus the next shuffling center can start his job while the previous shuffling center is still engaging in the process of proving. The correctness of the result is verified by the shuffling management center as soon as the shuffling center completes generating a proof. Therefore, at a same time, one shuffling center may be engaged in a shuffle-and-decrypt procedure, another shuffling center may be proving, and a shuffling management center may be verifying the proof reported from previous shuffling center. In this parallel scheduling, we measured 'Tally without proof' when the last shuffling center reports the result of his shuffle-and-decrypt and the shuffling management center verified the signature. For 10,000 voters, it was 69 seconds after the shuffling management center send the encrypted votes to the first shuffling center. The shuffling management center had to wait another 5.5 minutes to finish receiving the proofs from three shuffling centers and verifying them each.

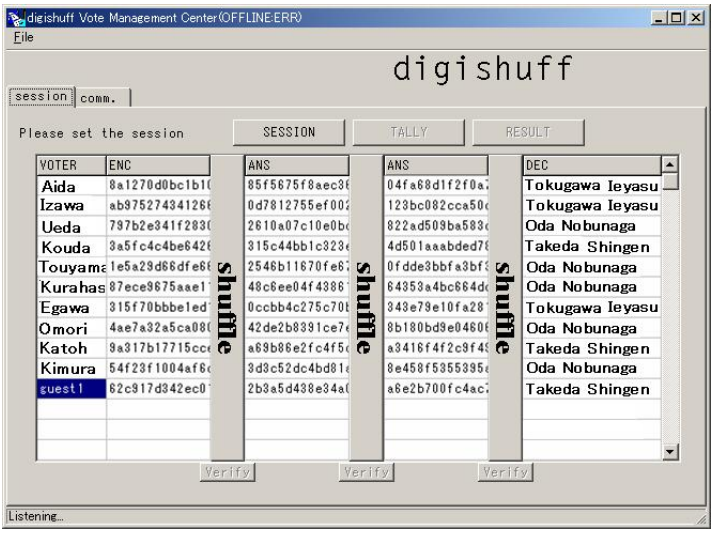

Fig. 2. An window of Tallying Result

The parallel scheduling is risky in the sense that if a shuffling center returns a properly signed wrong result, the following shuffling centers need to redo their job in order to obtain a correct tally. Redoing brings many threats. We nonetheless chose this implementation assuming such a fraud is unlikely to happen and even if it does, the shuffling center can be identified and be heavily blamed, Of course, the schedule can be always be changed to sequential one, so that the next shuffling receives the input only after the input has been verified, in cases where the speed is not of a question or the threat is large. In the sequential implementation, it takes 122 seconds for a shuffling center to shuffle-and-decrypt and generate its proof for 10,000 votes, and 105 seconds for a shuffling management center to verify the proof. We also note that 67 seconds out of 122 seconds needed for a shuffling center to perform its job can be done before he receives the input.

Figure 2 shows an example of tallying result collected at Shuffling Management Center.

5 Deployment in a Private Organization

The system described above has been used in a private organization with roughly 20,000 voters since 2004. The system is being used almost every other months to make organizational decisions reflecting opinions from anonymous members of the organization. The cases include election of the president of the organization, confirming the board member, changing the by-laws, and collecting the agreements from the members to exercise organizational rights. The organization is composed of multiple divisions. All the voting was tallied within the division.

The organization had user authentication infrastructure within their intranet. After the voter authorization protocol, the voter posted the encrypted vote generated with JAVA applet.

The administrators can chose the number of mixers in each voting. The secret keys of mixers are refreshed every time. There was no major trouble after 3 years of running the system but one, where a newly assigned administrator refreshed the secret key but announced the old public key by mistake. Since the old secret key had been properly destroyed, the voting with old public key had to revote.

It may be worth noting that the organization succeeded to cut the 90% of the cost they spend in paper-based voting. The cost cut is mostly due to the cut of man power to count paper ballots.

6 Ongoing Trial

We briefly describe here another ongoing trial using the components of the implemented system. The trial is to select young researcher award in a symposium, where roughly 650 participants of the symposium can vote among roughly 200 candidate presentations. The vote was conventionally done with paper ballots, where a voter pick the best five presentations, and write down the paper number, the title, and the presentator for each selected ones. A voter had to make a quick decision after he heard the last presentation and before he leaves the place. By introducing network voting system, it was easy for voters to search and chose the selection of his choice on web. The voters also had extra time for the selection because he can cast the ballot from his home.

Moreover, the voter is allowed to submit the vote many times, but only the latest one counts. Therefore, they could vote, say the third day of the four-days-symposium and can change the vote if there were better presentations in the last day.

7 Concluding Remarks

We discussed an implementation of mix-net based network voting scheme, which realizes many plausible features. We also discussed its actual use in binded voting within a private organization of voter size 20,000.

References

[Abe99] Abe, M.: Mix-networks on permutation networks. In: Lam, K.-Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 258–273. Springer, Heidelberg (1999)

[Br93] Brands, S.: An Efficient Off-line Electronic Cash System Based On The Representation Problem, CWI Technical Report CS-R9323 (1993)

[Cha81] Chaum, D.: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communications of the ACM, 84–88 (1981)

[CY85] Cohen, J.D., Yung, M.: Distributing the power of a government to enhance the privacy of voters. In: Annual Symposium on Principles of Distributed Computing, pp. 52–62 (1985)

[CGS97] Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. European Transactions on Telecommunications 8, 481–489 (1997); Preliminary version in: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 103–118. Springer, Heidelberg (1997)

[CFSY96] Cramer, R., Franklin, M., Schoenmakers, B., Yung, M.: Secure Secret Ballot Election Schemes with Linear Work. In: Advances in Cryptology — EUROCRYPT 1996 (1996)

[DJ01] Damgård, I., Jurik, M.: A Generalization, a Simplification and some Applications of Paillier's Probabilistic Public-Key system. In: Proc. of Public Key Cryptography 2001 (2001)

[E85] El Gamal, T.: A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithm. IEEE Transactions on Information Theory IT-31, 469–472 (1985)

[EVOX] `http://theory.lcs.mit.edu/~cis/voting/voting.html`

[FOO92] Fujioka, A., Okamoto, T., Ohta, K.: A Practical Secret Voting Scheme for Large Scale Elections. In: Zheng, Y., Seberry, J. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 244–251. Springer, Heidelberg (1993)

[FS86] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)

[F04] Furukawa, J.: Efficient, verifiable shuffle decryption and its requirement of unlinkability. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 319–332. Springer, Heidelberg (2004)

[FS01] Furukawa, J., Sako, K.: An Efficient Scheme for Proving a Shuffle. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 368–387. Springer, Heidelberg (2001)

[FMMOS02] Furukawa, J., Miyauchi, H., Mori, K., Obana, S., Sako, K.: An Implementation of a Universally Verifiable Electronic Voting Scheme based on Shuffling. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 16–30. Springer, Heidelberg (2003)

[MOV] Menezes, van Oorschot, Vanstone: Handbook of Applied Cryptography. CRC Press, Boca Raton

[Ne01] Neff, C.A.: A Verifiable Secret Shuffle and its Application to E-Voting. In: ACMCCS 2001, pp. 116–125 (2001)

[OMAFO] Ookubo, M., Miura, F., Abe, M., Fujioka, A., Okamoto, T.: An improvement of a practical secret voting scheme. In: Information Security Workshop 1999 (1999)

[PIK93] Park, C., Itoh, K., Kurosawa, K.: Efficient anonymous channel and all/Nothing election scheme. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 248–259. Springer, Heidelberg (1994)

[Sak94] Sako, K.: Electronic voting schemes allowing open objection to the tally. Transactions of IEICE E77-A (1) (Janaury 1994)

[SK94] Sako, K., Kilian, J.: Secure voting using partially compatible homomorphisms. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 411–424. Springer, Heidelberg (1994)

[SK95] Sako, K., Kilian, J.: Receipt-free mix-type voting scheme. In: Guillou, L.C., Quisquater, J.-J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 393–403. Springer, Heidelberg (1995)

[Schnorr] Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology 4, 161–174 (1991)

[SENSUS] `http://lorrie.cranor.org/voting/sensus/`