A Verifiable Secret Shuffle and its Application to E-Voting

C. Andrew Neff *

17 August, 2001

Abstract

We present a mathematical construct which provides a cryptographic protocol to *verifiably shuffle* a sequence of k modular integers, and discuss its application to secure, universally verifiable, multi-authority election schemes. The output of the shuffle operation is another sequence of k modular integers, each of which is the same secret power of a corresponding input element, but the order of elements in the output is kept secret. Though it is a trivial matter for the "shuffler" (who chooses the permutation of the elements to be applied) to compute the output from the input, the construction is important because it provides a linear size proof of correctness for the output sequence (i.e. a proof that it is of the form claimed) that can be checked by an arbitrary verifiers. The complexity of the protocol improves on that of Furukawa-Sako[16] both measured by number of exponentiations and by overall size.

The protocol is shown to be honest-verifier zeroknowledge in a special case, and is computational zeroknowledge in general. On the way to the final result, we also construct a generalization of the well known Chaum-Pedersen protocol for knowledge of discrete logarithm equality ([10], [7]). In fact, the generalization specializes *exactly* to the Chaum-Pedersen protocol in the case $k = 2$. This result may be of interest on its own.

An application to electronic voting is given that matches the features of the best current protocols with significant efficiency improvements. An alternative application to electronic voting is also given that introduces an entirely new paradigm for achieving *Universally Verifiable* elections.

Keywords: Electronic Voting, Universal Verifiability, Anonymous Credentials, Mix-net, Permutation, Verifiable Mix, Verifiable Shuffle, Honest-verifier, Zeroknowledge.

*aneff@votehere.net

1 Introduction

The notion of a shuffle of a collection of objects, records, or tokens is simple and intuitive, and useful examples abound in various daily human activities. A gambler in a casino knows that among the cards in his hand, each will be one of 52 unique values, and that no one else at the table will have duplicates of the ones he holds. He does not, however, have any knowledge of how the cards are distributed, even though he may have recorded the exact card order before they were shuffled by the dealer.

In the context of electronic data, the problem of achieving the same kind of random, yet verifiable permutation of an input sequence is surprisingly difficult. The problem is that the data itself is either always visible to the auditor, or it isn't. If it is, then the correspondence between input records and output records is trivial to reconstruct by the auditor, or other observer. If it isn't, then input and output records must be different representations of the same underlying data. But if the output is different enough (that is, encrypted well enough) that the auditor cannot reconstruct the correspondence, then how can the auditor be sure that the shuffler did not change the underlying data in the process of shuffling?

Most of the paper is devoted to giving an efficient (linear) method for solving this problem in an important context – ElGamal, or Diffie-Hellman encrypted data. In order to make the exposition as clear and concise as possible, the majority of the paper explicitly refers to the specific case where the operations are carried out in a prime subgroup of $\mathbf{Z}_p^*$, the multiplicative group of units modulo a large prime, p. However, the only properties of the underlying (multiplicative) group that we use is that the associated Diffie-Hellman problem is intractable. Thus, the shuffle protocol is also useful when the ElGamal cryptosystem is implemented over other groups such as elliptic curves.

The general Boolean proof techniques of [5] and [11] can also be used to construct a proof with the same properties, however, the resulting proof size (complexity) is quadratic, or worse, in the size of the input sequence.

The technique of this paper also offers several advantages over the cut-and-choose technique used in [26]. In this approach, the size of proof is dependent on the probability of a cheating prover that is required to satisfy all participants. In the shuffle protocol of this paper, this cheating probability is essentially k/q, where k is the number of elements to be shuffled, and q is the size of the subgroup of $\mathbf{Z}_p^*$ in which the elements are encrypted. Although no analysis of the proof size dependence on cheating probability is done in [26], it appears that, in order to obtain similarly low cheating probability, it will need to be orders of magnitude larger than the size of the proof given in this paper. (Moreover, if the [26] protocol is implemented non-interactively, the cheating probability would need to be chosen exceedingly small, because a malicious participant might use considerable off-line computation to generate a forged proof by exhaustive search. This of course, could be the case with the protocol of this paper as well, but the probability k/q is, for all practical values of k and q, certainly small enough – even for offline attacks.)

The results of this paper provide for several ways to implement a *universally verifiable* election protocol. Some of these are presented in the final sections. In this context, it is worth comparing the elegant homomorphic election protocol of [7]. That protocol works well when ballots have only questions of a simple *"choose (at most) m of n"* type. This effectively precludes *"write-in"* responses, as well as *"proportional type"* questions where the voter is expected to indicate answers in preferential order, and questions are tabulated in accordance with this preference. (Theoretically, proportional type questions can be handled by mapping each valid permutation of selections to a single yes/no response. However, in practice this is infeasible unless the number of choices is quite small.) A couple of somewhat less important disadvantages of the [7] scheme are that it expands vote data size considerably, and that it requires a voter *validity proof*. This proof further expands the vote data size by about an order of magnitude, and is unattractive from a practical perspective, because it presumes special purpose code to be running on the voter's computer.

The shuffle protocols are constructed entirely from elementary arithmetic operations. They are thus simple to implement, and are imminently practical for the anonymous credential application described.

1.1 Comparison to previous results

The number of exponentiations required to construct the proof is $8k+5$, where as the protocol of Furukawa-Sako[16] requires $18k+18$, which itself is a significant improvement over the roughly $642k$ exponentiations required by Sako-Kilian[26] and the $22k \log k$ exponentiations required by Abe-Hoshino[1]([2]). In the special case where the shuffler, or prover, knows the encrypted data, only $k+4$ exponentiations are required by the present protocol. The construction of this paper also has advantages when measured by size, or length, which is $8k+5$ modular integers, or group elements. In the case of $\mathbf{Z}_p$ implementation, most of these are "smaller" integers – typically 160 bits – though some of them are "larger" integers – typically 1024 bits. Furukawa-Sako[16] claim their proof size is $2^{11}k$ bits, and that the sizes for for Sako-Kilian[26] and Abe-Hoshino[1][2] respectively are $2^{18}k$ and $2^{14}k \log k$ bits. It is not clear at this time exactly how these size estimates compare with those of this paper, since they seem to have left out a dependence on the bit size of an integer. However, a rough count of the integers used in their protocol, seems to indicate more than $8k+5$.

All estimates of the efficiency of other papers are taken from [16]. However, in the case of [26] and [1]([2]) they may be conservative for a non-interactive implementation of the protocol.

1.2 Applications to voting

The voting application that occurs immediately is that which employs the usual tabulation/mixing center approach to provide anonymity. In this setting, the protocols of this paper offer important advantages. They are much more efficient, and allow the mixing centers to be completely independent of the authorities who hold some share of the key necessary to decrypt ballots.

Perhaps, however, a more valuable and exciting application of the new protocol is for creating "anonymous credentials". A member of an authorized group, identified only by a set of DSA, or Diffie-Hellman public keys, can authenticate group membership, and/or sign in a one time way, without revealing his/her *individual* identity. This leads to a novel solution to the voting problem that is universally verifiable, but does not require any special set of "authorities" in order to tabulate. It also offers a better privacy model to the voter, and speeds tabulation enormously since ballots do not need to be encrypted/decrypted. In effect, instead of mixing encrypted vote cyphertexts *after* ballots have been received at the vote collection center, voter credentials are mixed *before* the start of the election. This mixing can naturally be done by the voters themselves to achieve "anonymous authentication". (See section 6.1.) (It should be noted that the mixing could also be done by a set of authorities, thus providing a more efficient means to implement a threshold privacy election. One where, again, ballots do not need to be encrypted/decrypted.)

2 Notation

In the following, unless explicitly stated otherwise, n will be a positive integer, p and q will be prime integers, publicly known. Arithmetic operations are performed in the modular ring $\mathbf{Z}_p$ (or occasionally $\mathbf{Z}_n$), and $g \in \mathbf{Z}_p$ will have (prime) multiplicative order q. (So, trivially, $q \,|\, (p-1)$.) In each proof protocol, P will be the prover (shuffler) and V the verifier (auditor).

We recall the Chaum-Pedersen proof of equality for discrete logarithms. For $G, X, H, Y \in \mathbf{Z}_p$ this is a proof of knowledge for the relation

$$\log_G X = \log_H Y \tag{1}$$

It is not known to be zero-knowledge, however it is known to be honest-verifier zeroknowledge. In the next section, we will give a natural multi-variable generalization of this protocol which also has these properties. These are sufficient for our main application where the verifier is implemented via the Fiat-Shamir heuristic. (See [15] and [7].)

Definition 1 *An instance of this proof, as above, will be denoted by*

$$\mathcal{CP}\,(G,\, X,\, H,\, Y)\,.$$

Definition 2 *For fixed $g \in \mathbf{Z}_p^*$, let $\otimes_g$ be the binary operator on $\langle g \rangle \times \langle g \rangle$ defined by*

$$\log_g (x \otimes_g y) \;=\; \log_g x \, \log_g y$$

for all $x, y \in \langle g \rangle$. *Alternatively*

$$g^a \otimes_g g^b \;=\; g^{ab} \;=\; (g^a)^b \;=\; (g^b)^a$$

for all $a, b \in \mathbf{Z}_q$. *Following the conventions used for summations and multiplications, we also use the notation*

$$\bigotimes_{i=1}^{k}{}_g \, X_i \;=\; X_0 \otimes_g X_1 \otimes_g \cdots \otimes_g X_k$$

We refer to this operation as logarithmic multiplication base, g.

In each of the notations in the preceding definition, the subscript g may be omitted when its value is clear from context.

Remark 1 Notice that

$$\log_G X = \log_H Y \iff G \otimes_g Y = H \otimes_g X \tag{2}$$

We note the following collection of well know results since they will be heavily used in the remainder of the paper.

Lemma 1 *Let* $f(x) \in \mathbf{Z}_q[x]$, *be a polynomial of degree* d. *Then there are at most* d *values* $z_1, \ldots, z_d \in \mathbf{Z}_q$ *such that* $f(z_i) = 0$.

Corollary 1 *Let* $f(x)\,, g(x) \in \mathbf{Z}_q[x]$ *be two* monic *polynomials of degree at most* d, *with* $f \neq g$. *Then there are at most* $d-1$ *values* $z_1, \ldots, z_{d-1} \in \mathbf{Z}_q$ *such that* $f(z_i) = g(z_i)$.

Corollary 2 *Let* $f(x)\,, g(x) \in \mathbf{Z}_q[x]$ *be two* monic *polynomials of degree at most* d, *with* $f \neq g$. *If* $t \in_R \mathbf{Z}_q$ *(*t *is selected at random from* $\mathbf{Z}_q$*), then*

$$P(\{t : f(t) = g(t)\}) \;\leq\; \frac{d-1}{q}$$

Corollary 3 *Let* $f(x)\,, g(x) \in \mathbf{Z}_q[x]$ *be any two polynomials of degree at most* d. *Then for every constant* $R \neq 0$, *there are at most* d *values,* $z_1(R), \ldots, z_d(R)$, *such that* $f(z_i(R)) = R\, g(z_i(R))$.

Definition 3 *Let* $f(x)$ *be a polynomial in* $\mathbf{Z}_q[x]$. *We denote by* χ_f *the (unordered) set of all roots of* f.

$$\chi_f \;\doteq\; \{\, t \in \mathbf{Z}_q \;:\; f(t) = 0 \,\} \tag{3}$$

Definition 4 *If* $\Lambda \subset \mathbf{Z}_q$, *and* $R \in \mathbf{Z}_q$, *we write*

$$R\,\Lambda \;\doteq\; \{\, t \in \mathbf{Z}_q \;:\; t = Ru\,,\; u \in \Lambda \,\} \tag{4}$$

Corollary 4 *Let $f(x)\,,\, g(x) \in \mathbf{Z}_q[x]$ be any two polynomials of degree at most d. Fix constants, $R \neq 0$, $\gamma \neq 0$, and $\delta \neq 0$. If $t \in_R \mathbf{Z}_q$, then*

$$P(\{t : f(\gamma t) = R\, g(\delta t)\}) \;\leq\; \frac{d}{q}$$

Lemma 2 *Let $\mathbf{Z}_q^k$ be the standard k-dimensional vector space over $\mathbf{Z}_q$, and fix $v = (v_1, \ldots, \ldots v_k) \in \mathbf{Z}_q^k$, $v \neq 0$. If $r \in_R \mathbf{Z}_q^k$ is chosen at random, then*

$$P(\{r : v \cdot r = 0\}) \;=\; \frac{1}{q}$$

Definition 5 *Let $M = (m_{ij})$ be a $k \times l$ matrix. We denote the i^{th} row vector of M by $\rho_i(M)$ and the j^{th} column vector of M by $\tau_j(M)$. That is*

$$\rho_i(M) \;=\; (m_{i1}, \ldots, m_{il}) \tag{5}$$

$$\tau_j(M) \;=\; \begin{pmatrix} m_{1j} \\ \vdots \\ m_{kj} \end{pmatrix} \tag{6}$$

3 Proofs for iterated logarithmic multiplication

For the rest of this section, all logarithmic multiplications will be computed relative to a fixed element g, and hence we will omit the subscript in notation. The following problem is fundamental to the shuffle protocols which are to come later.

Iterated Logarithmic Multiplication Problem: Two sequences $\{X_i\}_{i=1}^k$ and $\{Y_i\}_{i=1}^k$ are publicly known. The prover, $\mathcal{P}$, also knows $u_i = \log_g X_i$ and $v_i = \log_g Y_i$ for all i, but these are unknown to the verifier, $\mathcal{V}$. $\mathcal{P}$ is required to convince $\mathcal{V}$of the relation

$$\bigotimes_{i=1}^{k} X_i \;=\; \bigotimes_{i=1}^{k} Y_i \tag{7}$$

without revealing any information about the secret logarithms u_i and v_i.

The protocol we give is precisely a higher dimensional generalization of the Chaum-Pedersen protocol discussed at the beginning of section 2. In fact, we will see that in the case $k = 2$, the protocol is *exactly* the Chaum-Pedersen protocol. The presentation will be considerably simplified by restricting the problem instance to a case where

$$X_i \neq 1 \;\;, Y_i \neq 1 \;\; \forall 1 \leq i \leq k \tag{8}$$

Clearly, if any of these inequalities do not hold, then there is no sense in constructing a proof since equation (7) can be seen to hold or not by inspection. (If $X_i = 1$ then $x_i = 0$ and so equation (7) holds if and only if $Y_j = 1$ for some j. Similarly with the roles of X and Y reversed.)

Iterated Logarithmic Multiplication Proof Protocol (ILMPP) :

1. $\mathcal{P}$ secretly generates, randomly and independently from $\mathbf{Z}_q$, $k-1$ elements, $\theta_1, \ldots \theta_{k-1}$. $\mathcal{P}$ then computes

$$\begin{aligned} A_1 &= Y_1^{\theta_1} \\ A_2 &= X_2^{\theta_1} Y_2^{\theta_2} \\ \vdots &= \vdots \\ A_i &= X_i^{\theta_{i-1}} Y_i^{\theta_i} \\ \vdots &= \vdots \\ A_k &= X_k^{\theta_{k-1}} \end{aligned} \tag{9}$$

   and reveals to $\mathcal{V}$ the sequence $A_1, \ldots, A_k$.

2. $\mathcal{V}$ generates a random challenge, $\gamma \in \mathbf{Z}_q$ and reveals it to $\mathcal{P}$.

3. $\mathcal{P}$ computes $k-1$ elements, $r_1, \ldots, r_{k-1}$, of $\mathbf{Z}_q$ satisfying

$$\begin{aligned} Y_1^{r_1} &= A_1 \, X_1^{-\gamma} \\ X_2^{r_1} Y_2^{r_2} &= A_2 \\ \vdots &= \vdots \\ X_i^{r_{i-1}} Y_i^{r_i} &= A_i \\ \vdots &= \vdots \\ X_k^{r_{k-1}} &= A_k \, Y_k^{(-1)^{(k-1)}\gamma} \end{aligned} \tag{10}$$

   and reveals the sequence $r_1, \ldots, r_{k-1}$ to $\mathcal{V}$. (We will see in the proof of completeness, below, how these values are computed.)

4. $\mathcal{V}$ accepts the proof if and only if all of the equations in (10) hold.

Theorem 1 *The ILMPP is a three-move, public coin proof of knowledge for the relationship in equation (7) which is special honest-verifier zeroknowledge. The number of exponentiations required to construct the proof is k, and the number of exponentiations required to verify it is $2k$. If $\mathcal{V}$ generates challenges randomly, the probability of a forged proof is $1/q$.*

Remark 2 Note that in constructing the proof, all exponentiations can be done to the same base, g, so fixed base algorithms can be employed. (See [18], p. 623.)

Proof: The protocol is clearly three-move and public coin. The exponentiation count in the construction of the proof looks like it should be $2k-2$, but actually it can be constructed with only k exponentiations. This is because $\mathcal{P}$ knows the logarithms x_i and y_i, and hence can compute A_i as $A_i = g^{\theta_{i-1}x_i + \theta_i y_i}$ for all $2 \leq i \leq k-1$.

Completeness

Completeness means that, given arbitrary $\vec{\theta} = (\theta_1, \ldots, \theta_{k-1})$ and γ, $\mathcal{P}$ can always find $\vec{r} = (r_1, \ldots, r_{k-1})$ satisfying the system of equations in (10). To see that this is the case, take $\log_g$ of each side of the equations in (10), and set $\bar{r}_i = r_i - \theta_i$ for $1 \leq i \leq k-1$. One obtains the following $k \times (k-1)$ system of linear equations in $\mathbf{Z}_q$ for $\bar{r}_1, \ldots, \bar{r}_{k-1}$

$$\begin{pmatrix} y_1 & 0 & 0 & 0 & \cdots & 0 \\ x_2 & y_2 & 0 & 0 & \cdots & 0 \\ 0 & x_3 & y_3 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & x_{k-1} & y_{k-1} \\ 0 & 0 & \cdots & 0 & 0 & x_k \end{pmatrix} \begin{pmatrix} \bar{r}_1 \\ \bar{r}_2 \\ \bar{r}_3 \\ \vdots \\ \bar{r}_{k-2} \\ \bar{r}_{k-1} \end{pmatrix} = \begin{pmatrix} -\gamma\, x_1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ (-1)^{(k-1)}\, \gamma\, y_k \end{pmatrix} \tag{11}$$

The $(k-1) \times (k-1)$ sub-system

$$\begin{pmatrix} x_2 & y_2 & 0 & 0 & \cdots & 0 \\ 0 & x_3 & y_3 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & x_{k-1} & y_{k-1} \\ 0 & 0 & \cdots & 0 & 0 & x_k \end{pmatrix} \begin{pmatrix} \bar{r}_1 \\ \bar{r}_2 \\ \vdots \\ \bar{r}_{k-2} \\ \bar{r}_{k-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \gamma\, y_k \end{pmatrix} \tag{12}$$

is non-singular since its determinant is $\prod_{i=2}^{k} x_i$, which is non-zero by assumption (8). Hence, one can always solve it for $\bar{r}_1, \ldots, \bar{r}_{k-1}$. In fact, the solution is

$$\bar{r}_i \;=\; (-1)^{(k-i-1)}\, \gamma \;\prod_{j=i+1}^{k} \left(\frac{y_j}{x_j}\right) \tag{13}$$

However, under the hypotheses of the problem, (12) actually implies (11). This is because

$$
\begin{aligned}
0 &= \prod_{i=1}^{k} x_i - \prod_{i-1}^{k} y_i \qquad (14)\\
&= \left| \begin{pmatrix}
x_1 & y_1 & 0 & 0 & 0 & \cdots & 0 \\
0 & x_2 & y_2 & 0 & 0 & \cdots & 0 \\
0 & 0 & x_3 & y_3 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\
0 & 0 & 0 & \cdots & 0 & x_{k-1} & y_{k-1} \\
(-1)^k y_k & 0 & 0 & \cdots & 0 & 0 & x_k
\end{pmatrix} \right|
\end{aligned}
$$

which, combined with the fact that the sub-matrix on the left of equation (12) is non-singular, means that the first column vector of the $k \times k$ matrix in (14) must be a linear combination of the remaining $k-1$ column vectors.

Soundness
If the first column vector of the matrix on the left of equation (14) is not a linear combination of the remaining $k-1$ column vectors, then there can be *at most* one value of $\gamma \in \mathbf{Z}_q$ for which equation (11) holds. Thus, if γ is chosen randomly, there is at most a chance of 1 in q that $\mathcal{P}$ can produce $r_1, \ldots, r_{k-1}$ which convince $\mathcal{V}$.

Special Honest-Verifier Zeroknowledge
Honest-verifier zero-knowledge holds because, for random γ and random $\vec{r} = (r_1, \ldots, r_{k-1})$, and for

$$
\vec{A} = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_{k-1} \\ A_k \end{pmatrix} = \begin{pmatrix} X_1^{\gamma} Y_1^{r_1} \\ X_2^{r_1} Y_2^{r_2} \\ \vdots \\ X_{k-1}^{r_{k-2}} Y_{k-1}^{r_{k-2}} \\ X_k^{r_{k-1}} Y_k^{(-1)^k \gamma} \end{pmatrix} \qquad (15)
$$

the triple $(\vec{A}, \gamma, r)$ is an accepting conversation. It is easy to see that the distribution so generated for $\vec{A}$ is identical to that generated according to (9), again because the first column vector of the matrix in (14) is a fixed linear combination of the remaining column vectors. So if $\mathcal{V}$ is honest, the simulation is perfect. Since the challenge, γ, can be chosen freely, we also have special honest-verifier zero-knowledge.

Remark 3 The solutions for $\bar{r}_i$ in (13) could also be written formally as

$$\bar{r}_i \;=\; (-1)^{(i-1)}\,\gamma\,\prod_{j=1}^{i}\left(\frac{x_j}{y_j}\right)$$

However, this will not work if some of the y_j are 0. In the case of equation (13), this problem was avoided by assumption (8). Of course, the main part of the solution could just have well be set up under the assumption that $y_i \neq 0$ for all $1 \leq i \leq k-1$ – the choice of expression for r_i just needs to be kept consistent with the form of the assumption.

Remark 4 We leave it to the reader to check that in the case $k = 2$, the SILMPP reduces exactly to the well known Chaum-Pedersen protocol. In so doing, it is worth recalling remark 1, equation (2).

Remark 5 Special Soundness
As is the case with the Chaum-Pedersen protocol, which proves that $\mathcal{P}$ knows $s = \log_G X = \log_H Y$, the SILMPP proves that $\mathcal{P}$ knows $s_1, \ldots, s_k$ such that $\log_g X_i = \log_g Y_i$, *and* $\prod_{i=1}^{k} s_i = 1$. This is clear because from two accepting conversations, $(\vec{A}, \gamma, \vec{r}\,)$ and $(\vec{A}, \gamma', \vec{r}\,')$, with the same first move and $\gamma \neq \gamma'$, a witness,

$$(w, \rho) \;=\; (\,\gamma - \gamma'\,,\,\vec{r} - \vec{r}\,'\,)$$

can be extracted satisfying

$$\begin{aligned} Y_1^{\rho_1} &= X_1^{-w} \\ Y_2^{\rho_2} &= X_2^{-\rho_1} \\ \vdots &= \vdots \\ Y_i^{\rho_i} &= X_i^{-\rho_{i-1}} \\ \vdots &= \vdots \\ Y_k^{(-1)^k\, w} &= X_k^{-\rho_{k-1}} \end{aligned} \tag{16}$$

Since $w = \gamma - \gamma' \neq 0$, it follows that $\rho_i \neq 0$ for all $1 \leq i \leq k-1$. Thus, with an appropriate small modification to the statement of the problem, it satisfies special soundness.

4 The Simple k-Shuffle

The first shuffle proof protocol we construct requires a restrictive set of conditions. It will be useful for two reasons. First, it is a basic building block of the more general shuffle proof protocol to come later. Fortuitously, it also serves a second important purpose. A single instance of this proof can be constructed to essentially "commit" a particular permutation. This can be important when

shuffles need to be performed on tuples of $\mathbf{Z}_p$ elements, which is exactly what is required when shuffling ElGamal pairs, as in the voting application.

Simple k-Shuffle Problem: Two sequences of k elements of $\mathbf{Z}_p$, $X_1, \ldots, X_k$, and $Y_1, \ldots, Y_k$ are publicly known. The prover, $\mathcal{P}$, also knows $x_i = \log_g X_i$ and $y_i = \log_g Y_i$, but these are unknown to the verifier, $\mathcal{V}$. In addition, constants c and d in $\mathbf{Z}_q$ are known only to $\mathcal{P}$, but commitments $C = g^c$ and $D = g^d$ are made public. $\mathcal{P}$ is required to convince $\mathcal{V}$ that there is some permutation, $\pi \in \Sigma_k$, with the property that

$$Y_i^d \;=\; X_{\pi(i)}^c \tag{17}$$

for all $1 \leq i \leq k$ without revealing any information about x_i, y_i, π, c, or d.

Remark 6 For this section, and the remainder of the paper, we will make the simplifying assumptions that in all shuffle constructions

1. $x_i \neq x_j$ for $i \neq j$ (and hence, of course, $y_i \neq y_j$ for $i \neq j$).
2. $x_i \neq 1$ for all $1 \leq i \leq k$.

There are obvious ways to handle these special cases. Moreover, in practice, they will "essentially never" occur since elements are usually random.

The protocol of the previous section, in combination with corollary 2, provide the tools necessary to solve this problem in a fairly straightforward manner.

Simple k-Shuffle Proof Protocol:

1. $\mathcal{V}$ generates a random $t \in \mathbf{Z}_q$ and gives it to $\mathcal{P}$ as a challenge.
2. $\mathcal{P}$ and $\mathcal{V}$ publicly compute $U = D^t = g^{dt}$, $W = C^t = g^{ct}$,

$$\vec{\hat{X}} = (\hat{X}_1, \ldots, \hat{X}_k) = (X_1/U, \ldots, X_k/U)$$

and

$$\vec{\hat{Y}} = (\hat{Y}_1, \ldots, \hat{Y}_k) = (Y_1/W, \ldots, Y_k/W)$$

3. $\mathcal{P}$ and $\mathcal{V}$ execute the SILPP for the two length $2k$ vectors

$$\begin{aligned} \Phi &= (\vec{\hat{X}}, \overbrace{C, C, \ldots, C}^{k}) \\ \Psi &= (\vec{\hat{Y}}, \overbrace{D, D, \ldots, D}^{k}) \end{aligned} \tag{18}$$

The protocol succeeds ($\mathcal{V}$ accepts the proof) if and only if $\mathcal{V}$ accepts this SILPP.

Theorem 2 *The Simple k-Shuffle Proof Protocol is a four-move, public coin proof of knowledge for the relationship in equation (17). It satisfies special soundness, and is special honest-verifier zeroknowledge. The number of exponentiations required to construct the proof is* $2k$*, and the number of exponentiations required to verify it is* $4k$*.*

If $\mathcal{V}$ *generates challenges randomly, the probability of a forged proof is less than or equal to*

$$(k-1)/q + (q-k+1)/q^2 = (qk-q+q-k+1)/q^2 < k/q$$

Remark 7 The observations of remark 2 also apply in this case.

Proof: All of the required properties follow immediately from the results of the previous section. (Special soundness can be argued from remark 5.) A forged proof can only be generated in two conditions.

1. The challenge t is one of the special values for which

$$\prod_{i=1}^{k}(t - x_i/d) \;=\; \prod_{i=1}^{k}(t - y_i/c)$$

2. The challenge t is not one of the special values in 1 above, *and* the SILMPP is forged.

By corollary 2, the probability of 1 is at most $(k-1)/q$, and the probability of 2 is $(q-k+1)/q^2$ by the results of the previous section.

4.1 A complexity improvement

Both the size and complexity of the simple k-shuffle protocol can be improved by a factor of 2. Instead of using corollary 2, we use corollary 4. Intuitively, we would like to replace the k copies of D and k copies of C in equation (18) with single entries g^{d^k} and g^{c^k} respectively. Unfortunately, this would ruin the zeroknowledge property of the protocol. Instead, we modify the protocol as follows.

Simple k-Shuffle Proof Protocol II:

1. $\mathcal{P}$ generates randomly and independently β from $\mathbf{Z}_q$ and τ from $\mathbf{Z}_q - \{0\}$, computes

$$\begin{aligned} B &= g^{\beta} \\ T &= g^{\tau} \end{aligned} \tag{19}$$

   and reveals B and T to $\mathcal{V}$.

2. $\mathcal{V}$ generates a random λ from $\mathbf{Z}_q$ and reveals it to $\mathcal{P}$.

3. $\mathcal{P}$ computes s by

$$s \;=\; \beta + \lambda\tau - \prod_{i=1}^{k}\left(\frac{x_i}{y_i}\right) \;=\; \beta + \lambda\tau - \left(\frac{d}{c}\right)^k \tag{20}$$

and reveals s to $\mathcal{V}$.

4. $\mathcal{V}$ generates a random $t \in \mathbf{Z}_q$ and gives it to $\mathcal{P}$ as a challenge.

5. $\mathcal{P}$ and $\mathcal{V}$ publicly compute $U = D^t = g^{dt}$, $W = C^t = g^{ct}$,

$$\vec{\hat{X}} = (\hat{X}_1, \ldots, \hat{X}_k) \;=\; (X_1/U, \ldots, X_k/U)$$

and

$$\vec{\hat{Y}} = (\hat{Y}_1, \ldots, \hat{Y}_k) \;=\; (Y_1/W, \ldots, Y_k/W)$$

6. $\mathcal{P}$ secretly generates, randomly and independently from $\mathbf{Z}_q$, k elements, $\theta_1, \ldots \theta_k$. $\mathcal{P}$ then computes

$$\begin{aligned}
A_1 &= \hat{Y}_1^{\theta_1} \\
A_2 &= \hat{X}_2^{\theta_1}\hat{Y}_2^{\theta_2} \\
&\vdots = \vdots \\
A_i &= \hat{X}_i^{\theta_{i-1}}\hat{Y}_i^{\theta_i} \\
&\vdots = \vdots \\
A_k &= \hat{X}_k^{\theta_{k-1}}\hat{Y}_k^{\theta_k} \\
A_{k+1} &= g^{\theta_k}
\end{aligned} \tag{21}$$

and reveals to $\mathcal{V}$ the sequence $A_1, \ldots, A_{k+1}$.

7. $\mathcal{V}$ generates a random challenge, $\gamma \in \mathbf{Z}_q$ and reveals it to $\mathcal{P}$.

8. $\mathcal{P}$ computes k elements, $r_1, \ldots, r_k$, of $\mathbf{Z}_q$ satisfying

$$\begin{aligned}
\hat{Y}_1^{r_1} &= A_1\,\hat{X}_1^{-\gamma} \\
\hat{X}_2^{r_1}\hat{Y}_2^{r_2} &= A_2 \\
&\vdots = \vdots \\
\hat{X}_i^{r_{i-1}}\hat{Y}_i^{r_i} &= A_i \\
&\vdots = \vdots \\
\hat{X}_k^{r_{k-1}}\hat{Y}_k^{r_k} &= A_k \\
g^{r_k} &= A_{k+1}\left(BT^{\lambda}g^{-s}\right)^{(-1)^k\gamma}
\end{aligned} \tag{22}$$

and reveals the sequence $r_1, \ldots, r_k$ to $\mathcal{V}$.

9. $\mathcal{V}$ accepts the proof if and only if all of the equations in (22) hold.

Theorem 3 *Simple k-Shuffle Proof Protocol II is a five-move, public coin proof of knowledge for the relationship in equation (17). It satisfies special soundness, and is special honest-verifier zeroknowledge. The number of exponentiations required to construct the proof is $k + 4$, and the number of exponentiations required to verify it is $2k + 2$.*

If $\mathcal{V}$ generates challenges randomly, the probability of a forged proof remains less than or equal to

$$(k-1)/q + (q-k+1)/q^2 = (qk - q + q - k + 1)/q^2 < k/q$$

Proof Sketch: All of the arguments are very similar, property by property, to the arguments constructed in the case of the original protocol. The main difference is that one makes an appeal to corollary 4 rather than corollary 2.

Full details of the proof will be included in a later version of this paper.

5 The General k-Shuffle

An obvious limitation of the simple k-Shuffle protocol is that the shuffler, $\mathcal{P}$, must know all the original exponents $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$. In many applications, this will not be the case. The goal of this section is to eliminate that restriction.

General k-Shuffle Problem: Two sequences of k elements of $\mathbf{Z}_p$, $X_1, \ldots, X_k$, and $Y_1, \ldots, Y_k$ are publicly known. In addition, a constant $c \in \mathbf{Z}_q$ is known only to $\mathcal{P}$, but commitments $C = g^c$ and $D = g^d$ are made public. $\mathcal{P}$ is required to convince $\mathcal{V}$ that there is some permutation, $\pi \in \Sigma_k$, with the property that

$$Y_i^d \;=\; X_{\pi(i)}^c \tag{23}$$

for all $1 \leq i \leq k$ without revealing any information about π, c, or d.

General k-Shuffle Proof Protocol: The protocol is constructed from a simple k-shuffle that has been "appropriately randomized" by the verifier, and an application of lemma 2. (To ease presentation, and avoid confusing notation, we will present the proof for the situation $d = 1$. It is a fairly straightforward matter to make the modifications for the general case.)

1. $\mathcal{P}$ chooses two sequences of k elements $e_{11}, \ldots, e_{1k}$, and $e_{21}, \ldots, e_{2k}$ from $\mathbf{Z}_q - \{0\}$. $\mathcal{P}$ also chooses a random, independent exponent d from $\mathbf{Z}_q - \{0\}$. $\mathcal{P}$ reveals to $\mathcal{V}$ the sequences $E_{1i} = g^{e_{1i}}$ and $E_{2i} = g^{e_{2i}}$.

2. $\mathcal{V}$ generates another two sequences $f_{11}, \ldots, f_{1k}$, and $f_{21}, \ldots, f_{2k}$ randomly and independently from $\mathbf{Z}_q - \{0\}$.

3. $\mathcal{P}$ computes the sequences

$$F_{ji} \;=\; g^{d\, f_{j\pi^{-1}(i)} e_{j\pi^{-1}(i)}} \quad 1 \le i \le k\ ,\ 1 \le j \le 2 \tag{24}$$

and reveals these to $\mathcal{V}$.

4. $\mathcal{V}$ generates a random challenge γ from $\mathbf{Z}_q$ and reveals it to $\mathcal{P}$.

5. $\mathcal{P}$ and $\mathcal{V}$ execute a simple k-shuffle protocol for the values $(\vec{U}, g, \vec{V}, D)$ where

$$\begin{aligned} U_i &= g^{f_{1i} e_{1i} + \gamma f_{2i} e_{2i}} \\ V_i &= g^{d\, f_{1\pi^{-1}(i)} e_{1\pi^{-1}(i)} + \gamma\, d\, f_{2\pi^{-1}(i)} e_{2\pi^{-1}(i)}} \\ D &= g^{d} \end{aligned} \tag{25}$$

(Notice that $\mathcal{P}$ need not explicitly compute U_i and V_i in order to construct the proof – only knowledge of the logarithms is required and these can be computed by simple multiplication and addition. When checking the proof, $\mathcal{V}$ must *compute* U_i and V_i as

$$\begin{aligned} U_i &= E_{1i}^{f_{1i}} E_{2i}^{\gamma f_{2i}} \\ V_i &= F_{1i} F_{2i}^{\gamma} \end{aligned} \tag{26}$$

on the other hand, $\mathcal{V}$ did not need to perform the exponentiations necessary to compute E_{ij} and F_{ij} in the first place.)

6. $\mathcal{P}$ generates random, independent $a_1, \ldots, a_k$ and $b_1, \ldots, b_{k-1}$ and sets

$$b_k \;=\; v_k^{-1} \left(\sum_{i=1}^{k-1} b_i v_i \;-\; d \sum_{i=1} a_i u_i \right) \tag{27}$$

where $u_i = \log U_i$ and $v_i = \log V_i$. $\mathcal{P}$ then reveals $A_i = g^{a_i}$ and $B_i = g^{b_i}$ to $\mathcal{V}$.

7. $\mathcal{V}$ generates a random challenge λ from $\mathbf{Z}_q$ and reveals it to $\mathcal{P}$.

8. $\mathcal{P}$ computes the exponents

$$\begin{aligned} s_i &= a_i + \lambda u_i \\ r_i &= b_i + \lambda v_i \end{aligned} \tag{28}$$

along with the quantities

$$\begin{aligned} P &= \prod_{i=1}^{k} X_i^{r_i} \\ Q &= \prod_{i=1}^{k} Y_i^{s_i} \end{aligned} \tag{29}$$

and reveals these to $\mathcal{V}$. (Note the places of r_i and s_i have been swapped from what might have been anticipated. Note also that $\mathcal{P}$ needs only $k+1$ exponentiations to compute P and Q since the ratio of their logs is known to be d.)

9. $\mathcal{P}$ and $\mathcal{V}$ execute a SILMPP (or Chaum-Pedersen proof) for the values (P, Q, D, C).

10. The protocol succeeds ($\mathcal{V}$ accepts) if and only if

    (a) $\mathcal{V}$ accepts the simple k-shuffle in step 5 (including the computation of U_i and V_i via equation (26)).

    (b) $\mathcal{V}$ accepts that for all $1 \leq i \leq k$

$$\begin{aligned} g^{s_i} &= A_i U_i^{\lambda} \\ g^{r_i} &= B_i V_i^{\lambda} \end{aligned} \tag{30}$$

    (c) $\mathcal{V}$ accepts the proof in step 9.

Theorem 4 *The general k-shuffle proof protocol is a seven-move, public coin proof of knowledge for the relationship in equation (23) which is computational zeroknowledge – that is, distinguishing between real and simulated proofs is as hard as the decision Diffie-Hellman problem. The number of exponentiations required to construct the proof is $8k + 5$, and the number of exponentiations required to verify it is $9k + 2$.*

If $\mathcal{V}$ generates challenges randomly, the probability of a forged proof is at most

$$\begin{aligned} & 1/q + (qk - k + 1)/q^2 + 1/q + 1/q \\ = \ & (qk + 5q - k + 1)/q^2 \ < \ (k + 4)/q \end{aligned} \tag{31}$$

Proof: The protocol is seven-move since the steps can be executed in parallel. (We have presented them sequentially for clarity.) It is obviously public coin.

Completeness and Soundness
Both of these properties follow immediately from lemma 2 and the corresponding properties of the simple k-shuffle in step 5.

Computational Zeroknowledge
A simulated proof is generated in much the same way as a real proof, the only difference is that in step 5, a simulator does not have knowledge of π. At this point, the simulator generates instead, U_i and V_i by the same rules as equation (25), except that a random permutation, σ, is chosen instead of π. The simple k-shuffle proof can be simulated, and the remaining steps can also be simulated in essentially the same way. A distinguisher can then distinguish

between real and simulated proofs only if it can distinguish between distributions of the form

$$\left((G_i)\ ;\ (G^d_{\pi(i)})\right)$$

and

$$\left((G_i)\ ;\ (G^d_{\sigma(i)})\right)$$

This problem can be reduced to the decision Diffie-Hellman problem by induction. Further, if these distributions can be distinguished, the original encryption of the shuffle is not sound.

6 k-Shuffles of DSA Public Keys

The general k-shuffle is ideally suited to verifiably permuting a set of DSA, or Diffie-Hellman public keys. By this we mean that a new set of DSA public keys is produced, which is computationally unlinkable to the original set, but which *verifiably* represents the same set of private keys. This can be extremely valuable when one wishes to anonymize a set of authenticated keys while still protecting the integrity of the original group of private keys – the election setting is just one such example.

We only sketch the technique here, but the details should then be completely obvious to the reader. It is assumed that initially all the public keys are of the form (g, H), $H = g^s$, where g is some fixed generator and s is the private key. That is, loosely, "all the keys use the same base". The protocol proceeds as follows:

1. Shuffler, or mixer, is presented with g and the list of keys H_i.
2. Shuffler executes the general k-shuffle with $C = g$, and $Y_i = H'_i$ (the new public keys), implementing the $\mathcal{V}$ via the Fiat-Shamir heuristic.
3. Shuffler "returns" the entire proof transcript.
4. Assuming the transcript verifies, set $g = C$, and $H_i = H'_i$. By changing the common base to C, the private keys all remain the same since

$$H = g^s \iff H' = C^s \tag{32}$$

6.1 Anonymous Voters

In the voting application, it is often said that for election integrity one must know "who voted", but for privacy, one *must not* know "how voted". The technique of this section solves the privacy/integrity dilemma in a new way. Instead of knowing "who voted", one only knows that the person who voted is *a member of a set of authorized voters!* As a consequence, we are left with a voting solution that

1. Does not require key sharing to implement a distributed trust tabulation scheme.

2. Guarantees *computational privacy* to the voter, rather than *threshold privacy*, which is a necessary evil of other voting solutions based on distributed trust tabulation. (If a threshold number of authorities agree to collude, all voters' votes can be decrypted.)

3. Does not require encryption or decryption of voted ballots.

Of course, one must look after the problem of "double voting", but the technique of this section is easily modified to take care of that as follows. (For simplicity, we describe the protocol as executed at vote time, with the voter playing the role of the shuffler, however, numerous obvious modifications exist allowing it to be executed separately.)

- In step 3, the voter (shuffler) – who knows one of the private keys s_0 in this case – *signs* his voted ballot using the DSA pair (s_0, H'_0) and base C of course. (H'_0 is the "post shuffle" public key which belongs to the voter. The voter knows its place in the new sequence, since he/she executed the shuffle.)

- In step 4, assuming that the shuffle transcript checks, and that the ballot signature checks, the vote center simply removes H'_0 from the list of authorized keys, and starts the process again waiting for the next ballot request. The new list of public keys is now one smaller, and unless the voter (shuffler) knew more than one private key in the first place, he/she now knows *none* of the new private keys, and hence can not vote again.

The resulting election protocol is *Universally Verifiable* if all the transcripts and signatures are maintained.

7 k-Shuffles of Tuples

It should be clear that in section 5, the simple shuffle generated essentially "froze" the permutation that could be proved. This makes it easy to see how to extend the previous section to shuffles of k *tuples* of elements of $\langle g \rangle$. Thinking of a sequence of k l-tuples as a $k \times l$ array, a single simple k shuffle can serve to prove that all columns have been permuted according to the same permutation. In particular, this extends to tuples of ElGamal pairs.

7.1 DSA key shuffles without common base

The observation of this section also allows a generalization of the DSA key shuffle protocol of section 6. Rather than maintaining the entire set of public keys to the same base, $g \leftrightarrow C$, the keys are maintained as independent pairs (g_i, H_i). The shuffler can pick an arbitrary subset of key pairs, (G_i, H_i), shuffle them "as

2-tuples", and return the result. This makes shuffling more manageable if the original set is large, at the cost of increasing the work per key by about 50%.

8 The Multi-Authority Voting Application

Much of the setting for the conventional voting application can be found in [7]. Votes are submitted as ElGamal pairs of the form $(g^{\alpha_i}, h^{\alpha_i} m)$ (or a sequence of these pairs if more data is required), where m is some standard encoding of the voter choices, the α_i are generated secretly by the voters, and h is a public parameter constructed via a dealerless secret sharing scheme ([21]). Once the polls are closed (voting finished), an independent collection of authorities sequentially shuffles the ballots. On output of the final shuffle, the final collection of encrypted ballots is decrypted in accordance with the threshold scheme, and the clear text votes are tabulated in full view by normal election rules.

The authorities who participate in the sequential shuffles, may be arbitrary in number, and they may be completely different from those who hold shares of the election private key. The sequence of ballots which are finally decrypted can only be matched with the original sequence of submitted ballots if *all* of the shuffling authorities collude, since each of their permutations is completely arbitrary.

Each shuffle is performed by an individual authority as follows:

1. β_i are chosen secretly, randomly and independently.
2. Each vote $v_i = (g^{\alpha_i}, h^{\alpha_i} m)$ is replaced, in sequence, by $(g^{\alpha_i+\beta_i}, h^{\alpha_i+\beta_i} m)$. A Chaum-Pedersen proof is published without revealing the secrets.
3. A shuffle with secret exponent c is performed on the resulting encrypted votes.
4. Step 1-2 are repeated.
5. At this point, the messages that are encrypted are the c-th power of the original messages. This is easily fixed by raising each coordinate of each vote to the $1/c$ power. A Chaum-Pedersen proof of this operation is equally easy to provide, thus keeping c secret while convincing verifiers, by simply reversing roles of g and $C = g^c$.

9 Conclusion

The protocols presented offer a practical method for performing shuffles, or mixes, of data and proving their correctness. They considerably improve the efficiency of previous methods both in theory and practice. The structured nature of the protocols lend themselves well to implementation. In a future version of this paper, we expect to improve further on the complexity and size.

We also believe the general shuffle proof can be modified to achieve honest-verifier zeroknowledge.

10 Acknowledgments

The author wishes to acknowledge the advice and wisdom of several important cryptographers who both motivated, and helped to improve this result. In random order, they are: Dan Boneh, Josh Benaloh, Moti Yung and Berry Schoenmakers. Without them, this paper might have been lost in the shuffle.

References

[1] M. Abe. *Mix-Networks on Permutation Networks - ASIACRYPT 99*, Lecture Notes in Computer Science, pp. 258-273, Springer-Verlag, 1999.

[2] M. Abe and F. Hoshino. Remarks on Mix-Network Based on Permutation Networks. *Proceedings 4th International Workshop on Practice and Theory in Public Key Cryptography PKC 2001*, Lecture Notes in Computer Science, pages 317-324, Springer-Verlag, 2001.

[3] J. Benaloh. Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret. *Advances in Cryptology - CRYPTO '86*, Lecture Notes in Computer Science, pp. 251-260, Springer-Verlag, Berlin, 1987.

[4] J. Benaloh, M. Yung. Distributing the power of a government to enhance the privacy of voters. *ACM Symposium on Principles of Distributed Computing*, pp. 52-62, 1986.

[5] R. Cramer, I. Damgrd, B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. *Advances in Cryptology - CRYPTO '94*, Lecture Notes in Computer Science, pp. 174-187, Springer-Verlag, Berlin, 1994.

[6] R. Cramer, M. Franklin, B. Schoenmakers, M. Yung. Multi-authority secret-ballot elections with linear work. *Advances in Cryptology - EUROCRYPT '96*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1996.

[7] R. Cramer, R. Gennaro, B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. *Advances in Cryptology - EUROCRYPT '97*, Lecture Notes in Computer Science, Springer-Verlag, 1997.

[8] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84-88, 1981.

[9] D. Chaum. Zero-knowledge undeniable signatures. *Advances in Cryptology - EUROCRYPT '90, Lecture Notes in Computer Science*, volume 473, pages 458-464, Springer-Verlag, 1991.

[10] D. Chaum and T.P. Pedersen. Wallet databases with observers. Advances in Cryptology - CRYPTO '92, volume 740 of *Lecture Notes in Compute Science*, pages 89-105, Berlin, 1993. Springer-Verlag.

[11] A. De Santis, G. Di Crescenzo, G. Persiano and M. Yung. On Monotone Formula Closure of SZK. *FOCS 94*, pp. 454-465.

[12] W. Diffie, M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644-654, 1976.

[13] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469-472, 1985.

[14] A. Fujioka, T. Okamoto, K. Ohta. A practical secret voting scheme for large scale elections. *Advances in Cryptology - AUSCRYPT '92*, Lecture Notes in Computer Science, pp. 244-251, Springer-Verlag, 1992.

[15] A. Fiat, A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. *Advances in Cryptology - CRYPTO '86*, Lecture Notes in Computer Science, pp. 186-194, Springer-Verlag, New York, 1987.

[16] J. Furukawa and K. Sako. An Efficient Scheme for Proving a Shuffle. To appear in *CRYPTO 2001*.

[17] R. Gennaro. Achieving independence efficiently and securely. *Proceedings 14th ACM Symposium on Principles of Distributed Computing (PODC '95)*, New York, 1995.

[18] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. Handbook of Applied Cryptography, CRC Press, 1997.

[19] N. Koblitz, A Course in Number Theory and Cryptography, 2nd edition, Springer, 1994.

[20] A. M. Odlyzko, Discrete logarithms in finite fields and their cryptographic significance, *Advances in Cryptology - EUROCRYPT '84*, Lecture Notes in Computer Science, Springer-Verlag, 1984.

[21] T. Pedersen. A threshold cryptosystem without a trusted party, *Advances in Cryptology - EUROCRYPT '91*, Lecture Notes in Computer Science, pp. 522-526, Springer-Verlag, 1991.

[22] C. Park, K. Itoh, K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. *Advances in Cryptology - EUROCRYPT '93*, Lecture Notes in Computer Science, pp. 248-259, Springer-Verlag, 1993.

[23] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161-174, 1991.

[24] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612-613, 1979.

[25] K. Sako, J. Kilian. Secure voting using partially compatible homomorphisms, *Advances in Cryptology - CRYPTO '94*, Lecture Notes in Computer Science, Springer-Verlag, 1994.

[26] K. Sako, J. Kilian. Receipt-free mix-type voting scheme – A practical solution to the implementation of a voting booth, *Advances in Cryptology - EUROCRYPT '95*, Lecture Notes in Computer Science, Springer-Verlag, 1995.

[27] J. Kilian, K. Sako, *Secure electronic voting using partially compatible homomorphisms*, awarded 2/27/1996, filed 8/19/1994.

[28] J. Kilian, K. Sako, *Secure anonymous message transfer and voting scheme*, awarded 10/28/1997, filed 1/23/1995.