Remarks on Mix-Network Based on Permutation Networks

Masayuki Abe and Fumitaka Hoshino

NTT Laboratories
Nippon Telegraph and Telephone Corporation
1-1 Hikari-no-oka, Yokosuka-shi, Kanagawa-ken, 239-0847 Japan
{abe,fhoshino}@isl.ntt.co.jp

Abstract. This paper addresses the security and efficiency issues of the Mix-net based on permutation networks introduced in [1]. We first show that the original construction results in a Mix-net that yields biased permutation, so it gives some advantage to adversaries. A simple repair is provided. We then observe that one of the original schemes can be improved so that the servers and verifier enjoy more efficient computation and communication.

1 Introduction

Secure networks will need to provide data integrity and authenticity, and existing networks like the Internet are pursuing these goals. In some applications, however, sending data in an anonymous way plays a central role. Anonymous voting, payments, or donations are typical examples of applications that concern user's privacy. Mix-net is a cryptographic technique that offers anonymity over non-anonymous, i.e., traceable networks; it hides the source of a message by mixing it with other message sources.

Since the notion of Mix-net was introduced by Chaum [3], much work has been done on providing more secure, efficient and widely applicable schemes [15,19,13,2,8,9,1,10,14]. Some of them are cryptoanalyzed and plausibly fixed [17,11,18,16,6,12].

In [1], Abe introduced an efficient construction of robust and publicly verifiable Mix-nets based on permutation networks. Since the resulting schemes provide $O(tN \log N)$ efficiency for N inputs and t tolerable corrupt mix-servers, they suit a small to moderate number of inputs. Those schemes are considered to be the most efficient ones that provide robustness and public verifiability.

This paper addresses the security and efficiency issues of the schemes in [1]. We first show that the construction results in a Mix-net that yields biased permutation, so an adversary can have more advantage in violating anonymity than random guessing. We provide a sure and simple solution that achieves uniform distribution over all permutations. We then observe that one of the original schemes can be improved so that the servers and a verifier enjoy more efficient computation and communication.

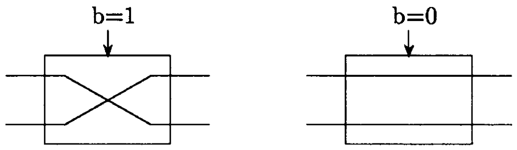

Fig. 1. Settings of a switching gate.

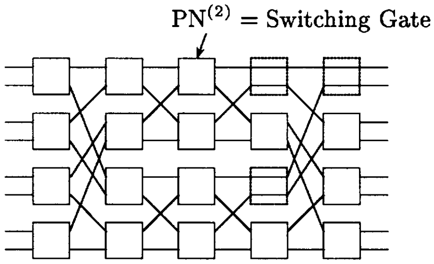

Fig. 2. PN$^{(8)}$ with PN$^{(4)}$

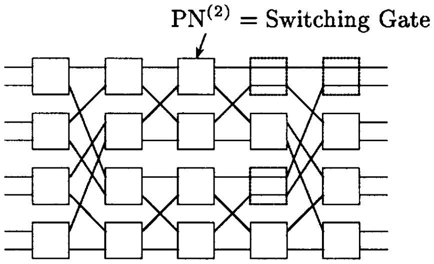

Fig. 3. PN$^{(8)}$ after decomposing PN$^{(4)}$

2 Review

We begin by reviewing the Benes permutation network [20]. Consider a switch that transposes two input signals according to a binary control signal as illustrated in Fig 1. The Benes permutation network is a network of such switches. It yields arbitrary permutation of the inputs by setting the switches. For N inputs, which is restricted to be a power of 2, $N \log_2 N - N + 1$ switches are necessary and sufficient to produce arbitrary permutation. Fig. 2 and 3 illustrate recursive construction of the Benes permutation network for 8 inputs, denoted by PN$^{(8)}$. Boxes represent switching gates, and the dotted ones indicate fixed gates that simply output the inputs.

Next we review the Mix-net in [1], which they call MiP-1. It consists of m servers. Up to $t(< m/2)$ of them can be malicious and colluding. Let p, q be primes. Let g be an element of Z_p^* that generates a prime subgroup of order q denoted by $\langle g \rangle$. Let x be the decryption key of the Mix-net and $y(= g^x)$ be the corresponding public key (all arithmetic operations in this paper are done in Z_p unless otherwise noted). The input to the Mix-net is a list of N ElGamal ciphertexts encrypted with y. An ElGamal ciphertext of message $msg \in \langle g \rangle$ is a pair (M, G) computed as $(M, G) = (msg \cdot y^s, g^s)$ where $s \in_R Z_q$. The servers logically simulate a series of $t+1$ permutation networks that take N ciphertexts as inputs. Let ξ denote the total number of columns in $t+1$ PN$^{(N)}$'s. That is, $\xi = (t+1)(2\log_2 N + 1)$. Each server is assigned some (or all) columns of switches in a permutation network in such a way that m servers are assigned ξ columns in total. Let $y_1, \ldots, y_\xi$ be shares of y such that $y = \prod_{i=1}^{\xi} y_i$. Let x_i be private key for y_i, i.e. $y_i = g^{x_i}$. The server that is in charge of the i-th column privately holds x_i. For the sake of robustness, x_i is shared among all servers by using $t+1$

threshold secret sharing. Let $\hat{y}_j$ denote $\prod_{i=j+1}^{\xi} y_i$. (Let $\hat{y}_0 = y$ and $\hat{y}_\xi = 1$.) At each switching gate in the j-th column, each input ciphertext (M, G) is partially decrypted and randomized as

$$(M', G') = (MG^{-x_j}\hat{y}_j^r, Gg^r)$$

for $r \in_U Z_q$, and output according to the *randomly selected* setting of the switch. (For all fixed gates and the switching gates in the ξ-th column, let $(M', G') = (MG^{-x_j}, G)$.) A zero-knowledge proof is then used to prove that the the outputs are correct. The proof can be done efficiently by using Chaum-Pedersen technique [4] combined with the OR-proof technique of [5]. Observe that (M', G') is an ElGamal ciphertext for public key $\hat{y}_j$. Accordingly, the outputs from the last column, which are $M' = MG^{-x_\xi}$, are the plaintexts that correspond to the input ciphertexts.

This scheme provides $O(tN \log N)$ efficiency and is known to best suit small numbers of inputs as the previous robust verifiable schemes, e.g., [19,13] require $O(mN\kappa)$ for error probability $2^{-\kappa}$.

It was proven in [1] that, under the decision Diffie-Hellman assumption, any poly-time adversary can guess the setting of each switch with probability only negligibly better than 1/2 since ElGamal encryption is indistinguishable and the proof is zero-knowledge with regard to switch setting. Hence, it was claimed that the scheme provides anonymity.

3 Security Issue

3.1 Biased Permutations

We show that randomly selecting each control signal results in a biased permutation even if all servers behave correctly. Let η be the number of switching gates in $\mathrm{PN}^{(N)}$, that is $\eta = N \log_2 N - N + 1$. Observe that $N! < 2^\eta$ holds for $N > 2$. So there are $2^\eta - N!$ permutations that have two or more different representations (switch settings). Accordingly, permutations with multiple representations are more likely to be generated than those with single representations if control signals are randomly set.

Indeed, the bias is even worse. Let ζ_N be the number of representations of $\mathrm{PN}^{(N)}$ for the identity permutation, which straightforwardly outputs the inputs. The identity permutation is clearly produced when $b = 0$ for all switching gates. Observe that the identity permutation can also be obtained by setting $b = 1$ for two gates located in the same row of the leftmost and rightmost columns such as gate 2 and 5 in Figure 2. Since there are $N/2 - 1$ such pairs of gates in $\mathrm{PN}^{(N)}$, we have

$$\zeta_N = {\zeta_{N/2}}^2 \cdot 2^{N/2-1}. \quad (1)$$

¿From the above recursive formula and the fact that $\zeta_2 = 1$, we obtain

$$\zeta_N = 2^{(N/2)\log_2(N/2) - N/2 + 1}. \quad (2)$$

Furthermore, as we will discuss in Section 3.2, there exist permutations that have only one representation. Thus, when each control signal is chosen randomly, the identity permutation appears with probability exponentially higher than the ones that have only one representation.

Table 1 shows the number of permutations that have more than two representations for $\mathrm{PN}^{(8)}$. It can be observed that some permutations are likely to be chosen with 32 times higher probability.

Table 1. Number of permutations in $PN^{(8)}$that have multiple switch settings. 128 of 8! permutations have 32 different switch settings.

#(equivalent settings)	#(permutations)
32	128
16	512
8	2816
5	2048
4	12288
2	14336
1	8192

3.2 Generating Non-biased Permutations

This bias can be eliminated in a simple way; first choose a permutation uniformly and then compute a proper setting of switches that represents the permutation. Clearly, this results in uniform distribution over all permutations. As in the original scheme, the setting of switches remains concealed.

A drawback is the increase of computation needed to transform a permutation to a switch setting. According to Waksman's algorithm [20], such a computation incurs $O(N \log N)$ time and memory.

The following description might be easier to follow if readers keep Figure 2 in mind. Let $I_1, \cdots, I_n$ be inputs to $\mathrm{PN}^{(n)}$ and $\tilde{O}_1, \cdots, \tilde{O}_n$ be the corresponding outputs from the switching gates in the first column of the permutation network. Similarly, let $O_1, \cdots, O_n$ be outputs of the network and $\tilde{I}_1, \cdots, \tilde{I}_n$ be the corresponding inputs to the switching gates in the last column. All switches are set to be straight, i.e. b=0, as the initial state.

The following algorithm sets the switching gates so that the network represents a given permutation $\sigma : \{1, \cdots, n\} \to \{1, \cdots, n\}$ that results in $O_i = I_{\sigma(i)}$ for $i = 1, \ldots, n$. Indeed, we describe the algorithm so that it sets the gates in only the first and last column of $\mathrm{PN}^{(n)}$, i.e., the numbered gates in Figure 2. Applying the algorithm repeatedly to the next inner set of gates, eventually sets all switching gates.

In the following, for some index j, $\bar{j}$ denotes the index such that I_j and $I_{\bar{j}}$ (or O_j and $O_{\bar{j}}$) are connected to the same gate. Let F_i be a flag, associated with O_i, which is initially unset. The algorithm starts with $i = 1$.

Step 1 Find the smallest i such that F_i is unset. If $i \neq 1$, arbitrarily set $b \in \{0,1\}$ for the switch O_i comes from.

Step 2 Repeat the following steps.

- **2-1** Set F_i.
- **2-2** Identify j such that $\tilde{I}_j = O_i$ by examining the switching gate that O_i comes from.
- **2-3** For k such that $I_k = \tilde{I}_j$ (i.e, $k = \sigma(i)$), if either j or k is even while the other is odd, then set $b = 1$ for the switch I_k enters.
- **2-4** Identify ℓ such that $\tilde{O}_\ell = I_{\bar{k}}$ by examining the switching gate $I_{\bar{k}}$ enters.
- **2-5** For t such that $O_t = \tilde{O}_\ell$ (i.e, $t = \sigma^{-1}(\bar{k})$), if either t or ℓ is even while the other is odd, then set $b = 1$ for the switch that O_t comes from.
- **2-6** Set F_t and let i be $\bar{t}$.
- **2-7** If F_i is already set, exit Step 2.

Step 3 Repeat Step 1-2 until all flags have been set.

The switches in the first and the last column are now configured and the internal permutation produced by the remaining switches can be computed from the current configuration. By recursively applying the above procedure to the remaining switches, taking the internal permutation as σ, one can configure all switches.

By using the above conversion algorithm, we can show that there exists a permutation with a unique representation. Observe that for any switch setting, there exists an execution of the above algorithm that results in the setting. Thus, the algorithm can yield all possible switch settings. Observe that Step 1 is deterministic when $i = 1$, and Step 2 is always deterministic. Thus, if a permutation has two or more representations, corresponding executions of the above algorithm must select different b at Step 1 in some level of recursion. Conversely, a permutation has a unique representation if Step 1 is executed only once in every recursive execution during the conversion. And clearly, one can manipulate the switch setting so that it happens.

4 Efficiency and Availability Issues

Here we observe that the original scheme reviewed in Section 2 unnecessarily uses many keys. In that scheme, decryption is incorporated into the task of *every* switching gate so that one layer of encryption is removed each time a ciphertext passes a gate. The following drawbacks arise.

- The servers have to securely maintain ξ private keys and m times many shares of the keys.
- The number of inputs has to be fixed before generating $y_1, \ldots, y_\xi$ as ξ depends on N.
- The zero-knowledge proof at each switching gate is expensive.

Our solution to eliminating these drawbacks is to incorporate the partial decryption only into the switching gates in the last column of each permutation network; the remaining switching gates perform randomization only such as

$(M', G') = (M\hat{y}_{i+1}^r, Gg^r)$. Although computational efficiency is asymptotically unchanged, this surely saves running time for computing factor G^{-x_i} in most switching gates and corresponding proofs. Furthermore, since the number of y_i becomes $t + 1$, which is independent of the number of inputs, the number of inputs need not be fixed before key generation.

5 Refined Scheme

The refined scheme, which reflects all the remarks in the previous sections, is given below.

[**Preparation**]
Let $\{1, \ldots, m\}$ denote servers. The servers agree on a subset of servers, say $W \subset \{1, \ldots, m\}$ that contains $t + 1$ servers.

[**Key Generation**]
The servers generate key pairs (y_i, x_i) for $i = 1, \ldots, t + 1$ by executing the key generation protocol of [7]. Public key y_i is published and private key x_i is shared in a threshold manner so that x_i can be re-constructed by $t + 1$ honest servers. Working server i, which refers to the i-th server listed in W, is given shares from other servers and computes x_i privately.

[**Mix processing**]
Let L_0 be a list of input ElGamal ciphertexts. Working server i takes list L_{i-1} and outputs list L_i. Working server i simulates the i-th permutation network in the following way.

1. Randomly choose a permutation $\pi : \{1, \ldots, N\} \to \{1, \ldots, N\}$.
2. Compute the switch setting that corresponds to π following the algorithm shown in Section 3.2.
3. For each switching gate except for the ones in the last column of the simulating permutation network, randomize each input ciphertext (M, G) as $(M', G') = (M\hat{y}_{i+1}^r, Gg^r)$ with random factor $r \in_R Z_q$. For each switching gate in the last column, perform randomization and decryption at the same time as $(M', G') = (MG^{-x_i}\hat{y}_i^r, Gg^r)$. (The last server only perform decryption as $M' = MG^{-x_{t+1}}$.) Then, output the resulting ciphertexts in order according to the switch setting defined in the previous step. For the fixed gate in the last column, decrypt the inputs as $(M', G') = (MG^{-x_i}, G)$ and simply output them. Then prove that the output is correct by using zero-knowledge proof. The proof protocol is unchanged from the original one in [1] (also see [10] for a slightly more efficient proof protocol).

All servers verify the proofs given by server i. If more than t servers agree that any of the proofs is faulty, then server i is disqualified. All servers then cooperatively reconstruct x_i and decrypt the ciphertexts in L_{i-1} in public.

The security of this scheme can be argued in the same way as shown in the original paper with the additional consideration that this scheme yields a uniform distribution over all permutations. (One missing argument in the security proof of [1] was about the distribution of the resulting permutation.)

Acknowledgment

The authors thank Tetsutaro Kobayashi for leading us to the security issue. Invaluable comments from Koutaro Suzuki are appreciated, too.

References

1. M. Abe. Mix-networks on permutation networks. In K. Lam, E. Okamoto, and C. Xing, editors, *Advances in Cryptology – Asiacrypt '99*, volume 1716 of *Lecture Notes in Computer Science*, pages 258–273. Springer-Verlag, 1999.
2. M. Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. *IEICE Transaction of Fundamentals of electronic Communications and Computer Science*, E83-A(7):1431–1440, July 2000. Presented at Eurocrypt'98.
3. D. L. Chaum. Untraceable electronic mail, return address, and digital pseudonyms. *Communications of the ACM*, 24:84–88, 1981.
4. D. L. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer-Verlag, 1993.
5. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994.
6. Y. Desmedt and K. Kurosawa. How to break a practical MIX and design a new one. In B. Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 557–572. Springer-Verlag, 2000.
7. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In J. Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310. Springer-Verlag, 1999.
8. M. Jakobsson. A practical mix. In K. Nyberg, editor, *Advances in Cryptology — EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 448–461. Springer-Verlag, 1998.
9. M. Jakobsson. Flash mixing. In *PODC99*, pages 83–89, 1999.
10. A. Juels and M. Jakobsson. Millimix. Technical Report 99-33, DIMACS Technical Report, June 1999.
11. M. Michels and P. Horster. Some remarks on a receipt-free and universally verifiable mix-type voting scheme. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 125–132. Springer-Verlag, 1996.
12. M. Mitomo and K. Kurosawa. Attack for flash MIX. In *Asiacrypt 2000* (to appear), 2000.
13. W. Ogata, K. Kurosawa, K. Sako, and K. Takatani. Fault tolerant anonymous channel. In *ICICS98*, volume 1334 of *Lecture Notes in Computer Science*, pages 440–444. Springer-Verlag, 1998.

14. M. Ohkubo and M. Abe. A length-invariant hybrid mix. In *Asiacrypt2000* (to appear), 2000.
15. C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. In T. Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259. Springer-Verlag, 1994.
16. B. Pfitzmann. Breaking an efficient anonymous channel. In A. D. Santis, editor, *Advances in Cryptology — EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 339–348. Springer-Verlag, 1995.
17. B. Pfitzmann and A. Pfitzmann. How to break the direct RSA implementation of MIXes. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology – Eurocrypt '89*, volume 434 of *Lecture Notes in Computer Science*, pages 373–381. Springer-Verlag, 1989.
18. K. Sako. An improved universally verifiable mix-type voting schemes. Unpublished Manuscript, 1995.
19. K. Sako and J. Kilian. Receipt-free mix-type voting scheme — a practical solution to the implementation of a voting booth —. In L. C. Guillou and J.-J. Quisquater, editors, *Advances in Cryptology — EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer-Verlag, 1995.
20. A. Waksman. A permutation network. *Journal of the Association for Computing Machinery*, 15(1):159–163, January 1968.