Flash Mixing

Markus Jakobsson
Information Sciences Research Center, Bell Labs
Murray Hill, New Jersey 07974
www.bell-labs.com/user/markusj

Abstract

By introducing novel methods for robust protocol design, to substitute for costly zero-knowledge schemes, we are able to produce a mixing scheme with significantly lower costs of operation than all previously known such schemes. The scheme takes a list of ElGamal encrypted messages, and produces as output a permuted list of encryptions of the same plaintext messages, such that corresponding items of the input and output cannot be correlated. For reasonably large inputs, the cost per participating server and element to be mixed is less than 200 modular multiplications, which is almost two orders of magnitude faster than the most efficient previously known method. The scheme has the novel feature of not requiring the knowledge of a secret key corresponding to the public key used to encrypt the messages constituting the input to the mix-network.

1 Introduction

The mix-network, a primitive for privacy, was introduced by Chaum [6] in 1981, and has recently been given a considerable amount of attention, both in terms of proposals for implementations, and as components of other schemes. Whereas the traditional use of mix-networks was limited to anonymizers [22] and election schemes [6, 11, 18, 20], it has recently also been used in payment schemes [14], in schemes detecting the corruption of secret signing keys [15], in telephony applications [10], and in protocols for efficient general multi-party computation [13]. In all of these schemes, its purpose is to provide communication privacy, i.e., to disassociate messages from their origin.

A mix network is a multi-party protocol constituting of a set of so-called mix servers, who take a list of ciphertexts and collectively produce and output a permuted list of items. Here, the output items are either the plaintexts that correspond to the input ciphertexts, or (as in this paper) ciphertexts that correspond to the same plaintexts as the input ciphertexts. The important functionality of a mix-network is that as long as at least one of the mix servers is honest, the privacy of the output is guaranteed. This means that it is infeasible for cheaters to match items of the output list to items of the input list with a probability non-negligibly exceeding that of guessing uniformly at random.

Each user that contributes ciphertexts to the input list needs to know only the encryption algorithm and the public key(s) associated with the mix-network. In this paper, we use standard ElGamal encryption, and participants contributing to the input list simply encrypt their messages using an ElGamal type public key associated with the mix-network. The input list is broadcast to all participating mix servers, who then together compute the output. All previously published mix-networks require the servers of the mix-network to know some secret keys relating to the public key used for encryption. In our current mix-scheme, it is sufficient that the mix servers know this public key (and that they all run the correct protocol.)

Early mix-networks (e.g., [6, 19, 22]) were not robust, i.e., it was possible for one or more of the participating servers to corrupt the computation without this being noticed, thereby causing an incorrect result to be produced and output. Clearly, this may, in turn, have repercussions for privacy for some schemes if it opens up to attacks in which related messages can be injected and identified using counting methods. Recently, several mix-networks (e.g., [1, 12, 17]) have implemented robustness, but at a high computational price. In general, in order for an end user to be able to trust the correctness of the output of a mix, it is necessary that all the participating servers provide a transferable proof that the output was correctly computed from the input, or that they establish this among themselves and then validate (using authentication methods) the generated list. For efficiency considerations, we take the latter approach in this paper, thereby requiring the end user of the mixed list to trust that at least one of the participating mix servers is honest.

The hitherto most efficient mix-network was proposed by Jakobsson [12], who observed that the high cost of other alternatives was largely due to the common use of costly zero-knowledge proofs to ensure correct computation. He posed it as an interesting research problem to reduce or eliminate the use of zero-knowledge protocols, in order to improve the efficiency of protocols. He therefore proposed an alternative paradigm for robustness, *repetition robustness*.

The mix-network suggested by Jakobsson [12] has two shortcomings, though. A first weakness is that it is possible to detect if two or more participants feed encryptions of the same plaintext to the mix-network, which in some instances (such as a yes/no vote) may be serious. Although this can be avoided by random padding of messages, it is still an

unpleasant property of a mix-network. A second drawback, although not a flaw *per se*, is the reliance on computationally heavy primitives such as exponentiation, and the (although limited) use of zero-knowledge protocols, keeping the costs of the scheme up.

For a reasonably-sized input (such as 2^{20} list items, and up), and for a negligible failure probability (chosen as 2^{-80}), Jakobsson's mix-network has a cost per input element and active server of approximately 80 exponentiations, each one of which costs about 200 multiplications for an exponent size of 160 bits, using standard window-based exponentiation techniques. This is low in comparison to the corresponding overhead of thousands of exponentiations incurred by cut-and-choose based protocols, such as [1, 17], but still expensive in comparison to the much lower overhead of the protocol we propose. For the same input size and security parameters, the computational cost of our protocol is approximately 185 multiplications per element and server, giving us an approximate speedup of a factor of 80 compared to the fastest known previous method. Our protocol does not have the weakness that encryptions of related plaintexts can be detected.

The lowering of the computational costs in the scheme we propose are made possible by the combination of new methods: (1) We shift from exponentiation to multiplication by the use of pre-processed re-encryption factors. These are generated using methods for addition chains. (2) We demonstrate a method to produce a *relative sorting* of several differently and randomly permuted lists, without disclosing the absolute permutation of any of these. This is a non-trivial task for lists that are permuted several times, given the non-commutative properties of permutation. (3) We demonstrate an inexpensive method for item-wise comparison of such relatively sorted lists, where all the items are probabilistically and independently encrypted. (4) We introduce the use of dummies in the list of ciphertexts to be mixed, in order to later allow the participants to trace these and verify that these were correctly manipulated. Given that these, as the other list items, previous to their tracing have been randomly re-encrypted and permuted, their correctness vouches (albeit probabilistically) for the correctness of the other list items.

In section 2, we start by introducing denotation, followed in section 3 by an informal presention of the system requirements. In section 4, we present the principles of our scheme. This is followed by an informal overview of the solution in section 5, immediately trailed by a more specific decription in section 6. After analysing the efficiency of our proposed scheme in section 7, we briefly state theorems governing the properties of our scheme in section 8. These are proven in the Appendix.

2 Preliminaries

Denotation. Onwards, all computation is performed modulo p, unless otherwise specified, where $p = kq + 1$, k is an integer, and both p and q are large primes. We let g be a generator of G_p.

Assumption. We make the assumption that the Decision Diffie-Hellman problem is hard. This means that if a string is selected uniformly at random with equal probability either as (g^a, g^b, g^{ab}) or (g^a, g^b, g^r) for $a, b, r \in_u Z_q$, then there does not exist a distinguishing adversary that given one of the above triples can determine from which one of the two distributions the triple is drawn, with a non-negligible advantage over a guess uniformly at random.

Encryption and re-encryption. An ElGamal encryption (a, b) of a message m w.r.t. a public key $y = g^x$ and a generator g is generated as $(a, b) = (my^\alpha, g^\alpha)$, for a random value $\alpha \in_u Z_q$. Given access to the secret key x, it can be decrypted by computing a/b^x. Such a pair (a, b) can be *re-encrypted* by computing and outputting (aY, bG), where we call $Y = y^\beta$ and $G = g^\beta$ the *re-encryption factors*, and $\beta \in_u Z_q$ the *re-encryption exponent*. Such a re-encrypted pair is homomorphic to the input ciphertext, i.e., the two ciphertexts correspond to the same plaintext. If (a_1, b_1) and (a_2, b_2) are both re-encryptions of (a, b), with re-encryption exponents β_1 resp. β_2, then we denote the value $\beta_1 - \beta_2 \, mod \, q$ the *relative re-encryption exponent*, and the corresponding values $y^{\beta_1-\beta_2}$ and $g^{\beta_1-\beta_2}$ the *relative re-encryption factors*. (Where in the above the first of the two new encryptions, that associated with β_1, is called the *reference*.) Finally, given two ciphertexts (a_1, b_1) and (a_2, b_2) we call the ciphertext (a_1a_2, b_1b_2) the *product* of the two former ciphertexts. The plaintext of this new ciphertext equals the product of the two plaintexts corresponding to the two input ciphertexts.

A tagging function. We let the function f be a keyed function that can be modelled by a random oracle. For simplicity, we assume that the range and the domain of f are equal but for a negligible fraction of values. We refer to [3, 2] for a treatment of random oracles and the use of keyed hash functions as random oracles.

3 Properties of our Mix

Participants. There are two types of participants, namely mix servers and decryption servers. This paper concentrates on the first type, which perform the mixing computation. The mix servers are assumed to be connected by a broadcast channel. Any set of the mix servers may perform the mixing, as this action does not require the knowledge of any secret key. However, for the result to be useful, it requires the decryption servers to approve and agree to use the result of the mix – which is correct if at least one mix participant is honest. Therefore, at least one mix server must be trusted by each decryption server.

Functionality. The *input* to the algorithm is a vector of ciphertexts $(E_1, \ldots E_N)$, where the vector item $E_i = (a_i, b_i)$ is an ElGamal encryption of a message m_i w.r.t. the public key y. The output is a permutation of a vector $(E'_1, \ldots E'_N)$, where E'_i and E_i are homomorphic, i.e., independent encryptions of the same plaintext. The elements of the output vector can later be straightforwardly decrypted, yielding a permutation of the plaintexts corresponding to the input ciphertexts.

Privacy. As long as one of the participating mix servers is honest, it will not be possible for any set of corrupt parties to correlate an item of the output string to its corresponding input item, with a probability non-negligibly better than a guess uniformly at random from all choices. This continues to hold if the plaintexts corresponding to the output ciphertexts are computed and published, or manipulated in any other manner, with the limitation of the possible choices to

all input ciphertexts whose plaintexts are not already known by any of the corrupt participants.

Robustness. As long as at least one mix server is honest, the output of the mix-network will be correct with an overwhelming probability.

4 Principles Employed

The largest portion of the computational cost of many multi-party public-key protocols is due to exponentiation, much of which is performed as part of zero-knowledge proofs. In this paper, we limit the costs of the mix scheme by limiting the use of zero-knowledge protocols, and of exponentiation in general.

In order to limit the use of costly zero-knowledge proofs, our solution derives its robustness from the principle of *repetition robustness* [12], the underlying idea of which is to repeat a chain of differently blinded/encrypted and permuted computations two or more times, after which the results are compared. If the results are equal, and only then, then the operation continues. Successful attacks must involve the modification of some chosen list items, and the same ones *for each copy* of the input list. Since the list items are independently and randomly re-encrypted in the different lists, and the lists are independently and randomly permuted, the probability of a successful attack can be made negligible in a setting with enough list elements and enough list copies. For reasonably-sized input lists, this method proves to be much less expensive than all known alternative methods.

In order to reduce costs we shift towards the use of multiplication instead of exponentiation. We avoid blinding of list items (as employed in [12]), and instead use re-encryption, whose computation costs are limited by the use of addition chains, e.g., [4, 5], or related methods.

Also to avoid the use of costly exponentiation, we introduce methods for comparing differently permuted and re-encrypted lists of data. These work by permuting a list of tags using the same permutations as previously used in the re-encryption phases, for each step applying a keyed one-way function to the tag elements. The tags will therefore be relative placeholders that do not reveal the actual permutations used, but only the *relative* permutations. Given these relative placeholders, lists of relative re-encryption exponents are compiled, where the result of the entire operation is a permuted list of relative exponents that describe how the different items, which are the outputs from the second re-encryption phase, were relatively re-encrypted. Here, too, the actual re-encryption exponents are not revealed. After re-encrypting the output lists of the second re-encryption phase using these relative re-encryption exponents, the resulting lists are sorted and compared. If the lists are homomorphic, but only then, then the output is valid with an overwhelming probability.

We also introduce a method for verifying consistency of computation in the mix-network. For this purpose we introduce two dummies in the list to be mixed. The two dummies are two (potentially identical) ciphertexts for which no true subset of servers know the plaintexts, nor the relationship to any input item. After the second re-encryption phase has ended, the servers reveal how the dummies were permuted. They also reveal the sum of all the re-encryption exponents corresponding to the input items and the first dummy (to verify that the product of these items does not change.) Finally, they reveal the re-encryption factor of the second dummy, whose purpose it is to decrease the probability of an attack in which the position of the first dummy is guessed, all the other elements altered by a multiplicative factor, and the first dummy altered to keep the product constant. With two dummies, *two* positions have to be correctly guessed – for each list copy – in order for the attack to succeed.

5 Overview and explanation of solution

Our protocol consists of the following steps:

1. *Generation and insertion of dummies.* Two dummies are generated and inserted in the list. The ciphertexts can either be constructed collectively, by contribution by each server of one portion of the ciphertext, or by setting the ciphertext to a pair of random elements of G_p that no subset of servers controls.

2. *Duplication.* $\tau \geq 2$ copies of the list are created, where $\tau = 1 - \frac{log_2 \epsilon}{2 log_2 N}$ for a security parameter ϵ indicating the maximum failure probability. Here, N is the number of elements of the input list.

3. *Generation of re-encryption factors.* Each mix server generates secret and random re-encryption factors. All exponentiation is performed using addition chains, in order to limit the cost of computation. (We note that this step may be performed during a pre-processing phase if desired.)

4. *First re-encryption.* In a serial action, each mix server individually re-encrypts each element of each of the lists given to him, and forwards random permutations of the resulting lists to the next server. This is performed to randomize the relative order of the list items for the second re-encryption phase, to prevent a set of servers at the beginning of the second re-encryption from mounting an attack made possible by them knowing these relative positions. Since at least one of the servers is assumed to be honest, the output lists have been randomly permuted and re-encrypted.

5. *Second re-encryption.* A similar re-encryption as above is performed, but with independent random values for re-encryption and permutation. This second step is employed to guarantee privacy of the result.

6. *Verifying first re-encryption.* The mix servers reveal the secret values used in the first re-encryption, and the computation is checked by the mix servers. This is done to prevent any server from manipulating the lists in the first re-encryption phase.

7. *Relative sorting and comparison of permuted lists.* The mix servers determine a relative sorting of all the permuted output lists (using methods soon to be detailed) and compare the resulting relatively sorted lists. This is done by determining a relative offset of the re-encryption factors used (as the same time as the relative sorting is determined) and re-encrypting each relatively sorted list using these offsets. If the results are identical, then the mix servers continue. This prevents with an overwhelming probability a set of attackers from successfully altering a portion of the lists, as they must for each copy guess what elements correspond to this portion and correctly make the same modification for each list.

8. *Verification of dummy values.* The positions of the first and the second dummy are determined by a selective trace-through of the mix. For each step of the second re-encryption phase, the server who performed the permutation and re-encryption publishes the sum (modulo q) of all re-encryption exponents used for all the elements *except for the second dummy.* Each server also publishes the re-encryption exponent used for the second dummy alone, and a description of how the two dummies were permuted. Then, he proves[1] that he knows the re-encryption factors employed for the first dummy (thereby substantiating that it was permuted as claimed.) All servers verify that the product of all input elements – except the second dummy – correspond to the product of the same output list. This is done by verifying the ciphertext corresponding to the product of all the input items except for the second dummy, re-encrypted using the published sum, results in a ciphertext that is the product of all the output elements – except for the second dummy again. (Note that it does not reveal this product, since the plaintexts of the dummies are unknown and uniformly distributed.) Finally, the re-encryption of the second dummy is verified. If no cheater was found, then the dummies are removed from the re-encrypted and permuted first list copy, and the resulting list output.

If cheating is detected during any step of the protocol, all the honest servers halt, and a cheater detection phase commences (in this phase, all secret random values are revealed, and the cheater(s), including those who will not cooperate, are pinpointed and replaced). We note that the plaintexts remain unrevealed. Afterwards, the protocol is restarted.

In the above description, we did not detail how the relative sorting and comparison is performed. An outline of the idea is as follows:

(a) Two types of lists are created, the so-called *tag list* and the so-called *offset list* (whose elements are denoted *tags* resp. *offsets*).

  (i) The first type of list, the tag list, contains $N+2$ unique elements in the domain of f. τ instances of this list are created, each one permuted according to the aggregate[2] permutation of the correspondingly numbered re-encryption list during the first re-encryption phase.

  (ii) The second type of list, the offset list, of which there are $\tau - 1$ copies (numbered $2 \ldots \tau$), contain the $N+2$ aggregate relative[3] re-encryption exponents used per element during the first re-encryption phase, permuted in the same manner as the tag list. (Thus, re-encrypting the output lists, numbered $2 \ldots \tau$, from the first re-encryption phase using these aggregate relative re-encryption exponents would render all the output lists of the first phase equal, but for their order. This is, however, not done.)

(b) Each mix server, in the same order as they performed the action during the second re-encryption phase, perform the following: First, he verifies that all the input tags of each list are different from each other (a multitude of methods spanning the spectrum of time vs. space trade-offs exist, we refer to [16] for a survey). Then, he applies the function f, keyed with a secret and random key only this server knows, to all the elements of all the tag lists. Next, he updates the elements of the offset lists by adding in the relative re-encryption exponents, where the difference is taken between items with identical tags, from the current list and the first list. Finally, he applies the same permutation to the updated lists as was applied by him during the second re-encryption phase, and passes the updated and permuted lists on to the next mix server.

(c) The two types of lists that are produced are a tagging indicating the relative order of the lists produced during the second re-encryption phase, resp. the relative re-encryption exponents of the list items, using the first list as their reference. The tag list is discarded (as it was only used to generate the correct offset-list by establishing the correct relative permutations). All the elements of the lists that constitute the output of the second re-encryption phase, but for those of the first list, are re-encrypted using the relative re-encryption exponents contained in the offset lists. (Again, the re-encryption factors are generated from the re-encryption exponents using addition chains.) Then, the lists are sorted and compared. If they are not identical then somebody must have cheated.

6 Solution Details

Our solution consists of the following steps:

1. *Generation and insertion of dummies.* Two dummies, (a_{N+1}, b_{N+1}) and (a_{N+2}, b_{N+2}), and are created in a way so that no subset of the servers know what plaintexts they correspond to, even if all the plaintexts corresponding to the input list were to be revealed.

2. *Duplication.* τ copies, L_{t0}, $1 \le t \le \tau$, of the appended input list L are created. Here, the index 0 denotes that no re-encryption has yet been performed.

3. *Generation of re-encryption factors.* Mix server j produces triples $(\alpha_{jti}, y^{\alpha_{jti}}, g^{\alpha_{jti}})$ and $(\beta_{jti}, y^{\beta_{jti}}, g^{\beta_{jti}})$, for $1 \le t \le \tau$, $1 \le i \le N+2$ and secret and random numbers $\alpha_{jti}, \beta_{jti} \in Z_q$. The computation is performed locally by each individual server, using methods for addition chains. (We refer to [4, 5] for a detailed description of these methods.)

4. *First re-encryption.* Mix server j takes as input the lists $L_{t(j-1)}$, for $1 \le t \le \tau$. He produces output lists L_{tj}, $1 \le t \le \tau$, where L_{tj} is a random and secret permutation Π_{tj} of the $N+2$ elements in the re-encrypted version of the list $L_{t(j-1)}$. Here, the re-encryption is performed element-wise, by computing the re-encrypted element (a_{jti}, b_{jti}). The result of the re-encryption is $(a_{(j-1)ti} y^{\alpha_{jti}}, b_{(j-1)ti} g^{\alpha_{jti}})$, using the precomputed re-encryption factors, $(y^{\alpha_{jti}}, g^{\alpha_{jti}})$. The output is given to the next mix server in the chain. The result of this step is denoted $L'_{t0} = L_{tk}$, for $1 \le t \le \tau$.

[1] This proof is performed using the scheme's only zero-knowledge proof, for which the verification proof for undeniable signatures [7, 8] may be used.

[2] With aggregate permutation, we mean the permutation with the same effect as the sequence of all the permutations performed.

[3] Each such relative re-encryption exponent is computed using the corresponding element of the first list as its reference.

5. *Second re-encryption.* This is performed as above, but for the input lists L'_{t0}, $1 \leq t \leq \tau$, and using new random and secret permutations Φ_{tj}, and the re-encryption factors $(y^{\beta_{jti}}, g^{\beta_{jti}})$. The result of this step is denoted $L''_{t0} = L'_{tk}$, for $1 \leq t \leq \tau$.

6. *Verifying first re-encryption.* Each mix server j reveals[4] the permutations Π_{tj}, and re-encryption factors α_{jti}, $1 \leq t \leq \tau$, $1 \leq i \leq N+2$. Each server computes the aggregate permutations $\Pi_t = \Pi_{t1} \circ \ldots \circ \Pi_{tk}$. Then, each server computes the aggregate re-encryption factors α_{ti}, which is the sum, modulo q, of the individial server-specific re-encryption factors *on the path of the permutation*. The servers compute[5] the pairs $(y^{\alpha_{ti}}, g^{\alpha_{ti}})$, for $1 \leq t \leq \tau$, $1 \leq i \leq N+2$. They then each verify that L''_{t0} is the permutation Π_t of the list L_{t0}, using re-encryption pair $(y^{\alpha_{ti}}, g^{\alpha_{ti}})$ for the ith element of L_{t0}. This is done plainly by performing said re-encryption and checking for equality.

7. *Relative sorting and comparison of permuted lists.* Using the previously detailed method, the lists are relatively sorted, and re-encrypted using the relative encryption exponents. The resulting lists are sorted and compared; if not all are identical, then somebody must have cheated.

8. *Verification of dummy values.* The position of and correctness of the two dummies are determined, as previously outlined, after which they are removed from the list, and the resulting first list output. The output is signed by all the cooperating mix servers to certify the correctness of the output.

7 Efficiency Analysis

We let $C(N)$ denote the cost of performing the computation of N exponentiations modulo p, with random exponents in Z_q and a common generator. Using the methods analyzed by Bleichenbacher [4] for addition chains, the *upper bound* of this cost (measured in the number of multiplications modulo p, for exponents in Z_q) for the N elements is shown to be

$$C(N) \leq log_2 q + \frac{(N-1)lnq}{ln(N-1) - lnln(N-1)}$$

Considering the cost in terms of the number of multiplications (which is the dominant cost) for the different protocol steps, we get the following numbers[6] per server: Step 1: 0 + Step 2: 0 + Step 3: $2C(2N\tau)$+ Step 4: $2N\tau$ + Step 5: $2N\tau$ + Step 6 and 7: $2C(2N\tau) + 4N\tau$+ Step 8: $N = 4C(2N\tau) + (8\tau+1)N$.

Given that $(\frac{1}{N+2}\frac{1}{N+1})^{\tau-1} < \frac{1}{N}^{2(\tau-1)} \leq \epsilon$ for a maximum failure probability of ϵ (this formula will be substantiated in the proof section), we have that $\tau - 1 \geq -\frac{log_2 \epsilon}{2log_2 N}$. Plugging in the values $\epsilon = 2^{-80}$, $N = 2^{20}$, we get $\tau = 3$ as the smallest possible value for the number of iterations. This in turn gives a total cost for the protocol, in the common case where no server cheats, of $4C(6 \times 2^{20}) + 25 \times 2^{20}$.

Plugging in values in the above formula, we get $C(6 \times 2^{20}) \approx 5.4 \times 10^7$, giving is an upper bound for the total cost of approximately 2.4×10^8 multiplications, or approximately 231 multiplications per element. Experimental results [4] give average costs of 80% or less of this upper bound, giving us an average cost of approximately 185 multiplications per server and input item.

8 Claims

The scheme is *correct*, i.e., if all the servers are honest then the correct output will be produced with an overwhelming probability. This follows trivially from the fact that the input elements will merely be permuted and re-encrypted as a result of the scheme.

As will be proven in Theorem 1, the scheme is *robust*, i.e., if a dishonest set of servers cause the wrong result to be computed, this will be detected with an overwhelming probability by the honest servers. Moreover, the latter will be able to determine what servers were corrupting the computation, and restart the same after having replaced the cheaters.

As will be proven in Theorem 2, the scheme satisfies *privacy*, i.e., as long as at least one server is honest, and the plaintexts of at least two input ciphertexts are unknown to the adversary, then the adversary has a non-negligible advantage compared to a uniform guess at random to determine the permutation performed on these unknown items from input to output, even if the plaintexts corresponding to the output items are revealed.

Acknowledgements

Many thanks to Daniel Bleichenbacher, Anand Desai and Ari Juels for important and valuable discussions, and to the program committee for valuable feedback leading to an improved final version.

References

[1] M. Abe, "Universally Verifiable Mix-net with Verification Work Independent of the Number of Mix-servers," Eurocrypt '98, pp. 437–447.

[2] M. Bellare, R. Canetti, H. Krawczyk, "Keying hash functions for message authentication," Crypto '96, pp. 1–15.

[3] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," First Annual Conference on Computer and Communications Security, ACM, 1993, pp. 62–73.

[4] D. Bleichenbacher, "Addition Chains for large sets," manuscript.

[5] J. Bos, M. Coster, "Addition Chain Heuristics," Crypto '89, pp. 400–407.

[6] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," Communications of the ACM, ACM 1981, pp. 84–88.

[4] If the random values to be revealed are generated as the output of a PRFG with a seed that is not used for any other random bit generation, this seed can be published in lieu of all the random values, in order to curb communication costs.

[5] This is done using addition chains. For maximum efficiency, this computation, along with the following verification, is postponed to be performed at the same time as the addition chain computation of the next step, in which lists are relatively sorted and compared. In the efficiency analysis, this performance enhancement is employed.

[6] For simplicity of denotation, we say that there were only N elements per list copy, thereby overlooking the two dummies. We also exclude the cost of the one zero-knowledge proof performed and verified per server, as this is a simple proof of valid exponentiation, whose cost is negligible for reasonably large inputs.

[7] D. Chaum, H. Van Antwerpen, "Undeniable Signatures," Crypto '89, pp. 212–216.

[8] D. Chaum, "Zero-Knowledge Undeniable Signatures," Eurocrypt '90, pp. 458–464.

[9] T. ElGamal "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," Crypto '84, pp. 10–18.

[10] H. Federrath, A. Pfitzmann, A. Jerichow "MIXes in Mobile Communication Systems: Location Management with Privacy," Workshop on Information Hiding, Cambridge (UK), Univ.of Cambridge, Isaac Newton Institute, 30.5.-1.6.96. Also `http://www.iig.uni-freiburg.de/dbskolleg/`

[11] A. Fujioka, T. Okamoto, K. Ohta, "A practical secret voting scheme for large scale elections," Auscrypt '92, pp. 244–251.

[12] M. Jakobsson, "A Practical Mix," Eurocrypt '98, pp. 448–461.

[13] M. Jakobsson, A. Juels, "Mix and Match: A New Approach to Secure Multiparty Computation," manuscript.

[14] M. Jakobsson, D. M'Raïhi, "Mix-based Electronic Payments," SAC '98.

[15] M. Jakobsson, J. Müller, "Improved Magic Ink Signatures Using Hints," Financial Cryptography '99.

[16] D. E. Knuth, "The Art of Computer Programming," Volume 2 – Seminumerical algorithms. Addison-Wesley, second edition, 1981.

[17] W. Ogata, K. Kurosawa, K. Sako, K. Takatani, "Fault Tolerant Anonymous Channel," ICICS '97, pp. 440–444.

[18] C. Park, K. Itoh, K. Kurosawa, "All/nothing election scheme and anonymous channel," Eurocrypt '93, pp. 248–259.

[19] A. Pfitzmann, B. Pfitzmann, M. Waidner, "ISDN-MIXes: Untraceable Communication with Very Small Bandwidth Overhead," Information Security, Proc. IFIP/Sec'91, May 1991, Brighton, D. T. Lindsay, W. L. Price (eds.), North-Holland, Amsterdam 1991, 245–258.

[20] K. Sako, J. Kilian, "Receipt-Free Mix-Type Voting Scheme," Eurocrypt '95, pp. 393–403.

[21] A. Shamir, "How to Share a Secret," Communications of the ACM, Vol. 22, 1979, pp. 612–613.

[22] P. Syverson, D. Goldschlag, M. Reed, "Anonymous connections and onion routing," IEEE Symposium on Security and Privacy, 1997.

[23] Y. Tsiounis, M. Yung, "On the security of ElGamal-based encryption," PKC '98.

A Proofs

Before stating and proving the two theorems, we outline and prove some lemmae that will be used for the theorems:

Lemma 1a: If the adversary can, with a non-negligible advantage ϵ over a guess uniformly at random, match any input to its corresponding output for the first or second re-encryption steps right after the completion of the respective step, then this adversarial strategy can be used as a black box to break the Decision Diffie-Hellman assumption with a probability $poly(\epsilon)$.

Proof of Lemma 1a: *(Sketch)*
It is well-known that the hardness of the Decision Diffie-Hellman problem is equivalent to the semantic security of ElGamal encryption. We assume that there is an adversarial strategy $\mathcal{A}$ that given the input vector to a mix step is able to match one input item to its corresponding output item (i.e., to an output item that corresponds to the same plaintext as the said input item) with a non-negligible andavntage ϵ over a guess uniformly at random. We then show how $\mathcal{A}$ can be used as a black box to determine with a probability $poly(\epsilon)$ whether a given ElGamal ciphertext (a, b) is an encryption of a given message m, which would mean that ElGamal encryption is *not* semantically secure, and therefore imply that the Decision Diffie-Hellman assumption does not hold. We will prove this by a standard diagonalization argument.
Our new adversary re-encrypts the input ciphertext (a, b) to randomize the query, and constructs $N+1$ other ciphertexts, whose plaintexts are different from m but chosen from the same distribution. He hands a permutation of these to the honest mix server. The output of the honest server is given to the black box $\mathcal{A}$, who, as assumed, matches one of the inputs to one of the outputs with a non-negligible advantage ϵ. This corresponds to the 0th test. Then, in the ith test, the adversary replaces the ith original output item (given some arbitrary ordering of these) by an encryption of a random and new message from the same distribution as m was chosen from. He then randomly re-encrypts and permutes all the resulting items, and runs $\mathcal{A}$ on the resulting vector. Clearly, $\mathcal{A}$ must with an overwhelming probability fail after the $N + 2$nd test, as by then, all the output items have been replaced and there is no correlation between input and output plaintexts. We assume, without loss of generality, that $\mathcal{A}$ loses his assumed non-negligible matching advantage after the kth test. By replacing the output item in this step by a random encryption of m, we can determine whether (a, b) corresponds to m or not. Namely, if $\mathcal{A}$ picks the "interesting" input item and fails in matching, then the replaced ciphertext was not an encryption of m. Similarly, if $\mathcal{A}$ picks the intersting input item with a probability non-negligibly different from $\frac{1}{N+2}$, this will be detected, and must mean that the the input and output vectors do not match. Otherwise, if he picks this intersting input item and succeeds in performing the match, this must be because the plaintext was not substituted by the replacement of the ciphertexts, i.e., (a, b) corresponds to m. Therefore, we will be able to distinguish the distributions with a non-negligible probability that is polynomially related to the adversary's advantage ϵ, which concludes the proof. □

Lemma 1b: If the adversary obtains a non-negligible advantage in matching any input to its corresponding output for the second re-encryption steps due to the execution of the step in which lists are relatively sorted, then this adversarial strategy can be used as a black box to distinguish the

output of the approximation of the random oracle f from a truly random oracle.

Proof of Lemma 1b: *(Sketch)*
In a similar argument as in Lemma 1a, we replace tag after tag by random numbers until the assumed adversarial strategy fails to match an input to an output item. At that point, we have – with a polynomial probability – distinguished the output of the f from a truly random number. □

Lemma 1c: An adverary cannot obtain a non-negligible advantage in matching any input to its corresponding output for the second re-encryption steps due to the execution of the step in which the dummies are verified.

Proof of Lemma 1c: *(Sketch)*
Herein, we modify the argument of Lemma 1a, so that instead of replacing an output item by a random ciphertext, we alter two of the output ciphertexts (but never the known second dummy) in a manner so that the product of all the elements, except for the second dummy, correspond to the same product of plaintexts (including that of the first dummy). By the same token, there can be no successful matching adversary in this setting. Now, we focus on what information is leaked by tracing the dummies. The position of the first dummy is proven using a zero-knowledge proof to show knowledge of the corresponding re-encryption exponent – this clearly can not leak any information given the fact that it can be simulated. Second, the product of all the ciphertexts except the second dummy cannot help, since this is constant, and does not give any information about either dummy value, or about the products of all the non-dummy items. Finally, the sum of the re-encryption exponents for all the elements to be multiplied together, and the re-encryption exponent of the second dummy, give no information about the permutation either, as they do not depend on the permutation (other than revealing the positions of the dummies, which is intended.) □

We are now ready to state and prove the theorems:

Theorem 1: The scheme is *robust*: If a dishonest set of servers cause the wrong result to be computed, this will be detected with probability $1 - \epsilon$ by the honest servers. Moreover, the latter will be able to determine what servers were corrupting the computation, and restart the same after having replaced the cheaters.

Proof of Theorem 1: *(Sketch)*
In lemma 1a, we determined that the adversary cannot guess the relative order of the items of the different list copies output from the first re-encryption step. This includes the positions of the dummies. Lemma 1a also determines that the adversary cannot guess the permutations performed during the second re-encryption phase. In the step in which the dummies are traced, each server has to prove to each other server how the dummies were permuted, and show (by exhibiting re-encryption exponents vs. sums of these) that the plaintext value of the second dummy was not altered, and that the product of the plaintext values of all the other elements were not altered. In order to successfully alter some elements in the final output, the adversary has to change elements included in this product so that the product is constant. Therefore, at least two elements have to be altered, none of which are the second dummy. In order for this not to be noticed in the step where lists are relatively sorted and compared, the adversary has to select the same two elements from each list copy. The probability of doing so is $(\frac{N+1}{N+2}\frac{N}{N+1})(\frac{N+1}{N+2}\frac{N}{N+1}\frac{1}{N+1}\frac{1}{N})^{\tau-1}$, where the first portion is the probability of not selecting the second dummy element in the first copy, and the second portion is to select the same two elements for the remaining $\tau - 1$ copies. This probability is smaller than $\frac{1}{N^{2(\tau-1)}}$, and by setting $\tau = 1 - \frac{log_2 \epsilon}{2log_2 N}$, as required, we see that the probability is smaller than the accepted failure probability ϵ. If any cheating is detected, all secret values relevant to the protocol are published. A cheater will therefore easily be detected by the honest servers re-performing his computation. Any server who does not reveal his secret values is branded a cheater. □

Theorem 2: The scheme satisfies *privacy* as long as at least one server is honest. If a polynomial-time adversary, to whom the plaintexts of at least two input ciphertexts are unknown, has a non-negligible advantage compared to a uniform guess at random to determine the permutation performed on these unknown items from input to output, then this adversarial strategy can be to break one of our two computational assumptions.

Proof of Theorem 2: *(Sketch)*
This follows automatically from our lemmae, which state that the adversary can only have a negligible advantage in matching inputs to outputs in any of the steps covered by these lemmae. In the case where no cheating is detected, which is the only case when a final output will be generated, the secret values used in the second re-encryption step will not be disclosed. Therefore, the theorem is found to hold. □