Information Leakage in Mix Networks with Randomized Partial Checking

Gedare Bloom* and Stefan Popoveniuc
George Washington University, 801 22nd St NW, Washington, DC 20052
E-mail: {gedare,poste}@gwu.edu, Tel: +1 (202) 994–7181, Fax: +1 (202) 994–4875
* Corresponding author

***Abstract*—We present a previously unknown vulnerability of mix networks (mix nets) that use pseudorandom permutations and that are audited with randomized partial checking (RPC). Our method relies on two simple observations: A mix that generates pseudorandom permutations only generates a limited subset of all possible permutations; in practical cases, RPC exposes enough information to uniquely identify the mix's permutation because of the gap between the number of permutations that the mix can generate and the total number of possible permutations. Exploiting this newly found vulnerability is difficult in practice. The only fix we see to this vulnerability is to maintain sufficient entropy used while generating the permutation. We are not aware of any applications using mix nets and RPC that can be exploited with this vulnerability.**

I. Introduction

Mix networks (mix nets) were introduced in 1981 by David Chaum [1] as one of the first constructions for protecting privacy in the digital world. Since then, mix nets have been a fertile ground for research in anonymous communications and applications. Mix nets provide a foundation for schemes in which privacy is of paramount importance, such as anonymous message delivery and electronic voting. The role of a mix net is to take a set of messages, or inputs, and (1) preserve the information in the messages while (2) shuffling their order to remove any *links* (correspondence between inputs and outputs). In other words, given the set of inputs and an output from the mix net, an adversary cannot identify the input from which the output came (with probability greater than a uniform random guess).

A mix net, or mix cascade, is a sequence of one or more servers called *mixes*. Each mix receives a batch of inputs from the previous mix, permutes and masks the inputs, and then sends the outputs to the next mix. Only the mix should know the correspondence between the inputs and the outputs. Data travels through the mix net serially, with each mix performing similar operations. The length of the mix net affects performance, reliability, and quality. More mixes improve the quality and reliability of the mix net, but the cost is reduced performance from replicating work.

The original mix nets, known as decryption mix nets (onion mixes or Chaumian mixes), are based on public key encryption. In this scheme, senders prepare an "onion" by successively encrypting the message, and some randomness, with the public keys of each mix in the order the mixes are used. As the onion travels through the mix net, each mix decrypts with its private key, removing one layer of encryption and the added randomness. The onions can be permuted by each mix sorting lexicographically the output decryptions, or by applying a secret random permutation to the decryptions.

Re-encryption mix nets use a public key encryption scheme, such as El Gamal, which allows re-encryption without knowledge of the private key in order to modify one layer of encryption. A message is sent through the mix net by encrypting with the first mix's public key and sending to that mix, which re-encrypts the message for the next mix net. The last mix in the mix net will decrypt to find the message. For a batch of input encryptions, the output of the mix is a set of re-encryptions, which are permuted by lexicographically sorting after re-encrypting.

In verifiable electronic voting systems, a mix net is used to de-correlate ballots from votes; that is, the inputs to the mix net are encrypted ballots and the outputs after mixing are the plaintext ballots. This system allows votes to be counted without revealing any voter's choice. In this situation, decryption mix nets need to use secret random permutations, because the client cannot be relied on to insert proper randomness in the onion. In particular, the machine that constructs the onions can choose the randomness so that the relative order of each message is preserved (or trivial/known) after each decryption. Re-encryption mix nets are still able to mix by sorting the re-encryptions. For an attacker to link encrypted ballots with plaintext ballots, the attacker must attack each mix's public key.

Recently, more specialized mix nets have been proposed, such as Punchscanian mixes [2] and pointer-based mixes [3]. These mix nets are based on commitment functions that might not be implemented using encryption, and therefore can not directly be attacked to correlate inputs with outputs. The commitments are hash outputs of mix information that can be revealed during the auditing phase. Such a system provides increased speed while mixing (no expensive public key encryptions).

Checking, or auditing, the correctness of a mix net means verifying that the input and output contain the same information. Two common auditing methods exist: zero knowledge proofs (ZKPs) [4] and randomized partial checking (RPC) [5]. Both methods are based on a challenge response mechanism and verify each mix individually. ZKPs require the mix to produce new information based on challenges, whereas RPC utilizes a simple observation to use existing information: the mixes are used serially, so some links for each mix can be revealed, as long as there is no path of revealed links from some input of the mix net to an output. RPC is faster than ZKPs [5], and RPC does not rely on producing new data, instead releasing data used in the mixing process.

Our work explores the possibility of finding all the links for each mix audited using RPC. If all links for a single mix can be found, then repeating the process for every mix will eventually reveal all links for the entire mix net. Our findings indicate that, when a mix's permutation is pseudorandom, RPC exposes much more than the links that are revealed explicitly. We show that, after checking a mix with RPC, all links can be accurately reconstructed such that the privacy guarantees of the mix net are violated. To reconstruct the links in practice currently requires more computational power than attacking the public key that obscures the mix net.

Our approach reveals a restriction on the number of possible permutations that can carry out the mixing operation, therefore the vulnerability exists for any of the mix net schemes with RPC applied. Throughout the rest of this paper, we focus on decryption mixes and secret random permutations. However, our analysis applies equally well to re-encryption mixes and to the newer non public-key mixing schemes.

The contribution of this work is to raise the issue that RPC plus the randomness used while mixing is a theoretical weakness, regardless of the mix net scheme that is used. If the mix net is a decryption mix net based on public keys, exploiting the weakness might be less efficient than attacking the public key encryption scheme. However, in re-encryption mix nets, and also mix nets based on commitments, this attack may offer a way for the attacker to find the entire permutation with less work than conventional attacks. In other words, this information leak provides a new attack vector for mix nets that cannot be attacked directly. This result primarily impacts designers of electronic voting systems, a number of which are based on mix nets [2], [6], [7].

A. Related Work

Much of the previous work in this area focuses on the relationship among multiple consecutive mixes in a mix net. In original RPC, two mixes can be paired so that the revealed outputs of the first are not revealed inputs of the second. However, now an adversary knows the unrevealed inputs of the first are the revealed outputs of the second, even if unable to distinguish elements within the set; thus the privacy set is cut in half for each pair of mixes. To fix this problem, Chaum [8] uses RPC across four consecutive mixes, with the first two mixes as before but the third mix reveals inputs corresponding to half the outputs of the second mix, and the fourth mix reveals only the unrevealed outputs of the third. Gomulkiewicz et al. [9] provide formal analysis of the information loss induced by Chaum's scheme with respect to the probability distribution that an input can be linked to an output. They find the connection between the inputs and outputs of a mix net are sufficiently random (assuming a good shuffle at each mix) to have assurance that the mix net meets the privacy demands of voting systems.

In our analysis, the way RPC is used with respect to surrounding mixes is irrelevant. We focus on a single mix at a time, and we are satisfied that half of all input-output pairs are linked. In this sense, we ignore information from surrounding mixes. We are unaware of other work that examines RPC from this point of view.

B. Useful Definitions

A *permutation* is a bijective function π with domain $1, 2, ..., n$ equal to the co-domain, and is a *random permutation* if for input i, $\pi(i)$ is selected uniformly at random from the co-domain. The entire permutation is given by the output sequence generated in order by $\pi(1), ..., \pi(n)$; each distinct ordering of the output sequence represents a distinct permutation. There are exactly $n!$ distinct permutations for domain of size n.

A permutation π of objects $X = x_1, x_2, ..., x_n$ is a reordering of the objects in which the object at input position i (x_i) *links* to the object at output position $\pi(i)$ ($x_{\pi(i)}$), that is x_i is moved to the $\pi(i)$'th position. A set of links contains link pairs of x_i and $x_{\pi(i)}$. We say that a permutation P *satisfies* a set $\mathcal{F}$ of links if, for every x_i linked to $x_{\pi(i)}$ in $\mathcal{F}$, x_i is linked to $x_{P(i)}$. In other words, the links in $\mathcal{F}$ of π are duplicated in P. For a set $\mathcal{F}$, $|\mathcal{F}|$ denotes the size of the set.

Information entropy [10] is a measure of the uncertainty (randomness) for a discrete random variable, and is calculated using the set of probabilities for the values (outcomes) of the variable. Formally, bit entropy H measures the uncertainty (in bits) within a set of probabilities $p_1, p_2, ..., p_n$ and is computed as $H = -\sum_{i=1}^{n} p_i \log_2(p_i)$. If x is a random variable that has values $x_1, x_2, ..., x_n$ occurring with probabilities $p(x_1), p(x_2), ..., p(x_n)$, then $H(x)$ means the value of H computed using the set of probabilities for each value of x. A fundamental example of bit entropy is that if x has uniform probability, then $H(x) = \log_2(n)$, and x has uncertainty of $\log_2(n)$ bits.

II. Permutation Generation

Consider the problem, for a single mix, of creating a permutation of n objects received as input, where the mix has access to at most k bits of randomness. Ideally, the mix would choose a permutation uniformly at random from the set of all permutations, and an attempt to guess the permutation chosen will be correct with probability $1/(n!)$. Unfortunately, this ideal situation is not practical.

To see the difficulty of achieving the ideal case, consider the bit entropy (H) required to have uncertainty (randomness) in the order of $n!$. H can be computed directly under the assumption that permutations are selected uniformly from a set of cardinality $n!$ as $H = \log_2(\frac{1}{(1/n!)}) = \log_2(n!)$ bits. From complexity theory $\log_2(n!)$ is in $O(n \log(n))$, which we derive with Stirling's approximation, which gives an upper bound for

$\log_2(n!)$:

$$\begin{aligned}\log_2(n!) &\approx \log_2\left(\sqrt{2\pi * n} * \frac{n^n}{e^n}\right)\\ &= \frac{1+\log_2(\pi)+\log_2(n)}{2} + n(\log_2(n) - \log_2(e)) .\end{aligned}$$

As n increases, the first term on the right approximates to $\log_2(n)/2$, while the second term gets arbitrarily close to $n\log_2(n)$: The number of bits needed to select a permutation grows faster than the number of objects permuted. If $n = 2^{10} = 1024$, then a mix would require about 10240 bits of randomness to select a random permutation. This may be infeasible, especially as n is allowed to grow arbitrarily large.

Instead, systems that use mix nets make use of some form of cryptographic, computationally secure, pseudorandom number generator (PRNG). PRNGs are used to generate a sequence of pseudorandom numbers from a *seed* of k bits. The PRNG preserves the entropy of the seed while generating more than k bits; of course, the sequence can never have more entropy than the seed, because no other source of randomness is consulted by the PRNG. Thus if a PRNG uses a seed of, for example, 128 bits, then the sequence of numbers generated also has only 128 bits of entropy, or randomness.

In re-encryption mix nets, each batch of inputs is re-encrypted with some randomness (a seed) and then sorted to produce a batch of mixed outputs. The seed and the mix's secret key determine the permutation. So the total entropy k of the permutation is the sum of the entropy of the seed and key.

In decryption mix nets, each layer of the onion is seeded when the onion is created. However, as noted previously, in electronic voting systems the onion's preparer is a voting maching that cannot be relied upon to insert true randomness. Therefore, decryption mix nets in voting use secret random permutations. The de facto standard for secret random permutations is to encrypt inputs with a block cipher, sort the encryptions, then decrypt and publish the outputs. Thus the encryption is never observed, only the shuffling. The block cipher's key (for fixed input) determines the permutation, so the total entropy k of the permutation is the entropy of the block cipher key.

If the mix only has access to k bits of entropy to generate a permutation, then any method used to select a random permutation is limited to k bits. Any such method can be equated to using a function that specifies a random lookup into a table of permutations, where the function can select up to 2^k indices of the table. This fact generalizes our results to any mix net scheme that uses pseudorandomness to create the permutation.

III. Analysis of Information Leak

A k-bit secret allows at most 2^k random permutations to be specified, but there are $n!$ permutations in total. The gap between $n!$ and 2^k is easily seen using Stirling's approximation, $\log_2(n!) \approx n\log_2(n)$. For any $n \geq k$,

$$n! \approx 2^{n\log_2(n)} \geq (2^k)^{\log_2(k)} > 2^k,$$

and the difference increases as n grows arbitrarily large while k is fixed. For example, a reasonable size might be $k = 256$, and with as few as $n = 64$ objects, $n! > 2^k$, so n does not even need to be as large as k for the gap to exist. In the remainder of this section, we demonstrate an information leak exploiting this inequality, and we discuss some ideas for reducing the ability of an adversary to exploit the leak.

A. Mixing Method and Assumptions

An entity M generates a permutation π of n objects for a single mix using one k-bit secret, and some publicly available parameters. At some point in time, M reveals a random subset $\mathcal{F}$ of links from its inputs to the permuted outputs. $|\mathcal{F}|$ is typically $n/2$, and the $|\mathcal{F}|$ pairs to reveal are chosen according to the rules for auditing the mix. The distribution on π is assumed to be uniform, and in practice this assumption is upheld by the cryptosystem used to generate π.

A common method of explicitly generating a permutation is to re-encrypt or decrypt the inputs and sort the outputs, as initially proposed by Chaum [1]. However, to be able to sort a vector of n objects, each object being potentially large, all the n objects must be kept in memory. If n is large, for example n is the number of ballots cast in a large national election, this becomes impractical. Thus implementers may choose to derive the permutation in another way, for example by using a public method that takes k bits as input, such as a PRNG. Our analysis focuses on the latter situation, but would work in either case.

Alternatively, the new types of mix nets based on commitments [2], [3] do not use any public key encryption mechanism. Such systems still derive their permutations in a deterministic manner from a fixed k bits of entropy, and so our analysis still applies. We consider these schemes as using secret random permutations for mixing.

B. The Leak

The mix's permutation π is chosen uniformly at random from a subset of 2^k permutations; we call this subset pos, for possible permutations. Figures 1a and 1b explore a simple example with $n = 4, k = 3$, where we have grouped together the permutations in pos; each dot corresponds to a distinct permutation on the n objects. Generating the entire pos set corresponds with generating all of the dots of the inner set of Figure 1a. Figure 1b leaves solid only the permutations that satisfy a fixed set $\mathcal{F}$ of $n/2 = 2$ links. We can readily verify that fixing two links leaves only two objects free to permute with each other.

In a perfect world, revealing $|\mathcal{F}|$ links would leave $n - |\mathcal{F}|$ links hidden, and there would be exactly $(n - |\mathcal{F}|)!$ permutations that satisfy $\mathcal{F}$. However, we have established that there are only 2^k permutations in pos.

Theorem 1 *The expected number of permutations satisfying a fixed subset $\mathcal{F} \subset pos$, where $|pos| = 2^k \leq n!$, is*

$$N(k, n, |\mathcal{F}|) = max\left(1, 2^k * \frac{(n-|\mathcal{F}|)!}{n!}\right) . \quad (1)$$

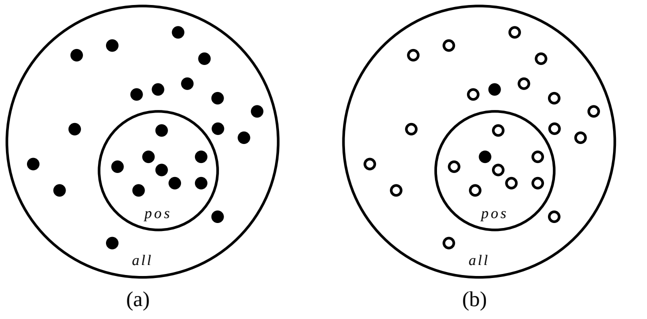

Fig. 1: A small example with $n = 4, k = 3$ and permutations in the pos (possible permutations) subset are grouped together. Fig. (b) shows that $(n/2)! = 2$ of the 24 permutations satisfy a set of links for exactly $n/2 = 2$ positions.

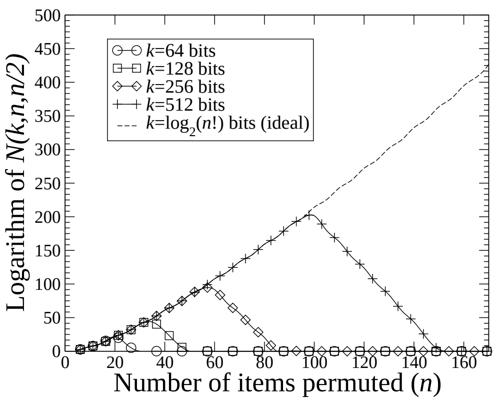

Fig. 2: Logarithm of Equation 1 for different values of k from 64 to 512, and for $k = log_2(n!)$ as the ideal case. Each plot represents a fixed k, and, when $2^k \leq n!$, there is linear decline in each of the plots (fewer permutations in pos satisfying $\mathcal{F}$). Increasing k increases the number of permutations in pos satisfying $\mathcal{F}$, but is dominated by the number of items being permuted. This shows that the more objects that get permuted, the greater amount of randomness is required to maintain ideal anonymity.

Proof: $N(k, n, |\mathcal{F}|)$ includes a lower bound because there is always at least 1 permutation (π) satisfying $\mathcal{F}$ in pos.

The 2^k permutations in pos are uniformly distributed across all $n!$ permutations. Thus the ratio of permutations that satisfy $\mathcal{F}$ within pos is equal to the ratio satisfying $\mathcal{F}$ in the whole set. Out of the $n!$ permutations, exactly $(n - |\mathcal{F}|)!$ satisfy $\mathcal{F}$, so the ratio is $\frac{(n-|\mathcal{F}|)!}{n!}$. This ratio is the same in pos, and multiplying by $|pos|$ yields equation 1. ■

Figure 1b demonstrates Equation 1 for a simple example, and Figure 2 shows a logarithmic plot of the equation.

Equation 1 is simple, but does not readily lend itself to analysis or intuitive comparisons. Again using Stirling's approximation and substituting $|\mathcal{F}| = n/f$ we find

$$\begin{aligned}
2^k * \frac{(n-|\mathcal{F}|)!}{n!} &= 2^{k+\log_2((n-n/f)!)-\log_2(n!)} \\
&\approx 2^{k+(n-n/f)\log_2(n-n/f)-n\log_2(n)} \\
&= 2^{k+(n-n/f)\log_2(1-1/f)-(n/f)\log_2(n)}, \\
\text{which, for } f &= 2 \text{ (the common case of } |\mathcal{F}| = n/2\text{), is} \\
&= 2^{k-(n/2)(1+\log_2(n))} .
\end{aligned}$$

If $k < (n/2)(1 + \log_2(n))$, then $2^k * \frac{(n-|\mathcal{F}|)!}{n!}$ is less than 1 and $N(k, n, |\mathcal{F}|) = 1$. From the plots in Figure 2, we see that maintaining multiple permutations in pos satisfying $\mathcal{F}$ requires more than linear growth in k with respect to n; this is also obvious from the approximation, but was not immediately obvious from Equation 1.

The probability that a permutation $P \in pos$ satisfying $\mathcal{F}$ is π is 1 divided by the number of permutations in pos that satisfy $\mathcal{F}$. Also, for fixed k and n, as $|\mathcal{F}|$ increases Equation 1 decreases toward 1. Thus, revealing more links decreases the number of permutations in pos that satisfy those links and increases the probability that there is only one permutation in pos that will satisfy the links.

C. Discussion

We do not know of any polynomial-time exploits, however we do provide an explanation for what an unbounded adversary can do with knowledge of the information leak. Suppose a computationally unbounded adversary A mounts a brute force attack by trying every k-bit secret. By using all 2^k possible secrets, A creates a table of all permutations in pos. From the analysis above, the probability that exactly one permutation in pos satisfies $\mathcal{F}$ rapidly approaches 1. Therefore A can determine the permutation π chosen by M. Note that, in the case of decryption mix nets at least, such an attacker would have an easier job attacking the encryption keys used. In a computationally bounded setting, exploiting this leak is equivalent to cryptanalyzing state of the art cryptosystems, and is not an immediate concern.

If an electronic voting system was exploitable through this vulnerability, then the secrecy of the ballot would be violated. Mix nets are used to provide anonymization of ballots, by transforming encrypted ballots as inputs in to plaintext votes as outputs. The encrypted ballots contain some identifying information, which allows a voter to match their receipt with the encrypted ballot and verify the ballot's integrity. The mix net iteratively shuffles and decrypts these ballots, until finally the vote is revealed with no other identifying information; thus a voter cannot give proof of a particular vote. However, if an adversary is able to link the encrypted ballots with the plaintext votes, the voter's privacy is lost.

In systems that use RPC, if the PRNG is computationally secure, then the information leak is not exploitable by a computationally bound adversary. System designers can vary the parameters of function N, given in Equation 1, so that everlasting privacy is guaranteed in an information theoretic setting. For example, the plot shown in Figure 2 can be generated for any $|\mathcal{F}|$ to find how much entropy (k) is necessary for any given number of n items to permute. We also approximated Equation 1 in terms of $|\mathcal{F}| = n/f$. This approximation provides a reference point for making analyses in situations different from the common case of $f = 2$. Using Stirling's approximation provides an upper bound, so the approximation is conservative and safe, if somewhat loose.

IV. Conclusion

We analyzed RPC for verifying the correctness of a mix net and conclude that this particular method of correctness checking may open the door to an attack that results in linking the inputs with the outputs for the entire mix net. Current approaches to generating permutations basically use cryptographic PRNGs to extend a seed of k bits to a longer sequence with the same amount of entropy as the seed. This limited randomness, when combined with RPC, theoretically can allow an attacker to link all inputs and outputs of each individual mix, leading to privacy loss for the mix net. To provide information theoretic privacy, any pseudo-random strategy should use $n \log_2(n)$ truly random bits in the seed. Existing cryptographic PRNGs make use of contemporary block ciphers and hash functions, which provide sufficient defense against modern attack methods.

In our analysis, exploiting the leak is currently impractical; however, the vulnerability exists and cryptanalysis might make a computationally bound exploit possible. Given enough computational power, a motivated attacker can apparently link every input of a single mix to every output, without breaking the public encryption key. This is a surprising result that is not intuitive to current understanding of the effect of revealing mix input-output pairs. For commitment based mix nets that use RPC, this result provides an attack vector where none was previously known to exist.

Our key result is that revealing half of the input-output pairs of a permutation leads, with high probability, to exactly one permutation with those pairs. Any application that reveals links in a random permutation generated from a cryptographic PRNG (for example, mix net-based electronic voting systems audited using RPC) is relevant to our analysis. System designers can avoid practical exploits of this vulnerability by maintaining a cryptographically secure amount of randomness used for deriving the permutation, so that attacking the mechanism of permutation generation is as hard as breaking modern cryptographic primitives.

We expect that most implementations are already secure with respect to current computing power and computational security of cryptographic PRNGs, but this vulnerability is important to discuss and analyze further. Also of interest would be examining the current techniques used for generating permutations to determine if this theoretical vulnerability can be exploited in a practical manner.

Acknowledgments

The authors thank Poorvi Vora and her research group for providing stimulating discussion and for pointing out ways to make this paper clearer and more concise. We also thank the anonymous reviewers. This work was partially supported by NSF Grant CNS-0934725 and CNS-0831149.

References

[1] D. L. Chaum, "Untraceable electronic mail, return address, and digital pseudonym," *Communication of ACM*, February 1981.

[2] S. Popoveniuc and B. Hosp, "An introduction to PunchScan," in *IAVoSS Workshop On Trustworthy Elections (WOTE 2006)*. Robinson College, Cambridge UK: http://punchscan.org/papers/popoveniuc_hosp_punchscan_introduction.pdf, June 2006.

[3] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. Sherman, and P. Vora, "Scantegrity: End-to-End Voter-Verifiable optical- scan voting," *Security & Privacy, IEEE*, vol. 6, no. 3, pp. 40–46, 2008.

[4] C. A. Neff, "A verifiable secret shuffle and its application to e-voting," in *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*. New York, NY, USA: ACM, 2001, pp. 116–125.

[5] M. Jakobsson, A. Juels, and R. L. Rivest, "Making mix nets robust for electronic voting by randomized partial checking," in *Proceedings of the 11th USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2002, pp. 339–353.

[6] D. Chaum, J. van de Graaf, P. Y. A. Ryan, and P. L. Vora, "Secret ballot elections with unconditional integrity," Cryptology ePrint Archive, Report 2007/270, IACR Eprint, Tech. Rep., 2007, http://eprint.iacr.org/ or http://www.seas.gwu.edu/~poorvi/cgrv2007.pdf.

[7] D. Chaum, P. Y. A. Ryan, and S. Schneider, "A practical voter-verifiable election scheme," in *In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, ESORICS, volume 3679 of Lecture Notes in Computer Science*. Springer, 2005, pp. 118–139.

[8] D. Chaum, "Secret-ballot receipts: True voter-verifiable elections," *IEEE Security & Privacy*, vol. 2, no. 1, pp. 38–47, 2004.

[9] M. Gomulkiewicz, M. Klonowski, and M. Kutylowski, "Rapid mixing and security of chaum's visual electronic voting," in *ESORICS*, ser. Lecture Notes in Computer Science, E. Snekkenes and D. Gollmann, Eds., vol. 2808. Springer, 2003, pp. 132–145.

[10] C. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, July, October 1948, http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf.