University of Wroclaw
Faculty of Mathematics and Computer Science
Institute of Mathematics

Paweł Dietrich

Improving Randomized Partial Checking

Bachelor dissertation
written under supervision of
Filip Zagórski, PhD

Wrocław 2022

Abstract

Randomized Partial Checking (RPC) [6] is a protocol proposed by Jakobsson, Juels and Rivest to validate operations performed by mix networks. Mix networks are often used in voting schemes to anonymize ballots. RPC is used to publicly verify if ballots are decrypted correctly. RPC was a building block of many voting schemes: Prêt à Voter [2], Civitas [5], Scantegrity II [3] as well as voting-systems used in real-world elections (*e.g.,* in Australia [1]).

In [6] it was argued that when k entries are modified by a mix server it would be detected by RPC with probability:

$$1 - 2^{-k}.$$

Khazaei with Wikström [7] and later Küsters, Truderung and Vogt [8] noticed that security guarantees of RPC scheme are off. The true probability of detecting manipulation of k elements is only:

$$1 - \left(\frac{3}{4}\right)^k.$$

In this thesis we improve chances of detecting a mix server's misbehavior. When k errors are made, the proposed RPC finds them with probability of $1 - 2^{-k}$ as it was initially promised.

The proposed version of the protocol introduces minimal changes and does not publish additional commitments. In conclusion, with the improved protocol the elections results can be more accurate, meaning this version is more likely to find any error than the original version was.

1 Introduction

In the following sections we are going to introduce mix networks and show how the original Randomized Rartial Rhecking (RPC) works.

1.1 Mix networks

Mix networks on an abstract level are black box objects having n inputs and n outputs. They perform two tasks at once. As the name suggests one of the tasks is shuffling the inputs, the other one is processing them. In particular a single stage of a mix network can permute and decrypt n cryptograms, which were given as an input, providing n permuted messages as the output. One can join multiple stages as long as it makes sense to process the output of the previous stage further.

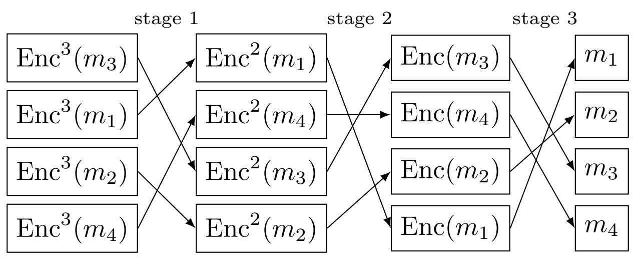

Figure 1: An example 3-stage decryption mix network.

Among other applications, a mix network can be used where a need for forgetting the origin of data arises. Chaum [4] suggests using mix servers for anonymizing senders of e-mail messages. Others have suggested using a mix network to anonymize votes cast during an election. This would enable election organizers to publish anonymized ballots and allow external verification of the election results.

One (boring) way to simulate a single stage mix network is to establish a trusted party who follows the protocol:

Protocol 1 Mix–network

1. Publish a public key pk for encrypting inputs and keep a private key sk for decrypting inputs. Any asymmetrical cipher can be used.
2. Wait for all inputs $\langle c_1, \ldots, c_n \rangle$ to be delivered.
3. Establish a permutation π, apply it on the inputs getting $\langle c_{\pi(1)}, \ldots, c_{\pi(n)} \rangle$ and forget π.
4. Decrypt all the mixed inputs and return them $\langle \mathrm{Dec}_{sk}(c_{\pi(1)}), \ldots, \mathrm{Dec}_{sk}(c_{\pi(n)}) \rangle$.

This straight forward protocol is fine as long as all users trust the mixer. When a mix network follows the protocol, its input-output relation remains secret and it would not violate the ballot secrecy. Not only that, its output is just a permutation of its input, which implies that the set of collected ballots and the set of counted ballots are equal.

Unfortunately in a real election it is usually impossible to find such a trusted party. Not only a nefarious mixer can reveal parts of the input-output relation, but also return anything instead of a properly decrypted input. Of course the details of a particular attack depend on the operation/cipher used and meaning of the processed data.

The two presented threats can be addressed. When joining many stages, while each stage being operated by a different party's server, we can fix the secrecy problem. Firstly, if any of the parties keeps its input-output relation secret, the whole mix input-output relation is secret. From the perspective of the voter his ballot secrecy is safe as long as he trusts any of the mixing parties in following the protocol. On the other hand, proving a mix input-output relation is indeed a permutation seems to be much more difficult. At first it can seem impossible, as revealing the permutation will void the secrecy. Randomized Partial Checking of a mix network is a method of verifying a permutation without revealing it.

1.2 Randomized Partial Checking

The precise definition is available in the original paper [6] by Jakobsson, Juels and Rivest. For our purposes the simplified description below will be sufficient.

Let us redefine how the permutation on each stage is established. Instead of randomly selecting one permutation π, we randomly select two (π_L and π_R), then compose them ($\pi = \pi_R \circ \pi_L$). This gives us the benefit of having a non-publishable substage[1], while not altering the distribution of the mixer's choices.

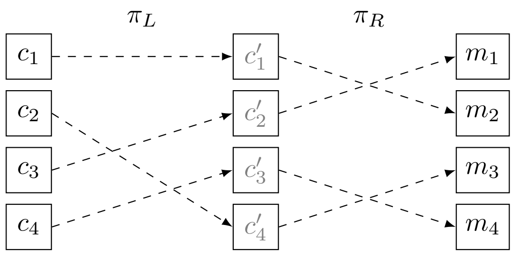

Figure 2: The left and right substages.

The protocol does not change much. Firstly, we do the mixing like previously, but the mixer is required to publish mixing artifacts in form of commitments – more on them later. After mixing and showing outputs and commitments the verification starts. The verifier provides a mixer with n bits. Each bit determines which commitment will be checked.

[1]a dual way to visualize it is to give each party two adjacent stages to operate without publishing the data between them

Protocol 2 Original RPC

1. Generate a key pair. Publish a public key pk for encrypting inputs and keep a secret key sk for decrypting inputs.
2. Wait for all inputs $\langle c_1, c_2, \ldots, c_n \rangle$ to be delivered.
3. Establish a permutations π_L and π_R, apply them on the inputs while decrypting them.

$$\langle c'_1, \ldots, c'_n \rangle = \left\langle \mathrm{Dec}_{sk}\left(c_{\pi_L(1)}\right), \mathrm{Dec}_{sk}\left(c_{\pi_L(2)}\right), \ldots, \mathrm{Dec}_{sk}\left(c_{\pi_L(n)}\right) \right\rangle$$

$$\langle m_1, \ldots, m_n \rangle = \left\langle \mathrm{Dec}_{sk}\left(c'_{\pi_R(1)}\right), \mathrm{Dec}_{sk}\left(c'_{\pi_R(2)}\right), \ldots, \mathrm{Dec}_{sk}\left(c'_{\pi_R(n)}\right) \right\rangle$$

4. Publish the output $\langle m_1, \ldots, m_n \rangle$ and the commitments: $C_L(i)$ and $C_R(i)$ for every i.
5. Wait for the challenge bits $\langle b_1, b_2, \ldots, b_n \rangle$. Depending on them open appropriate commitments and show zero–knowledge proofs of the correct decryption.

Commitments

A stage should publish $2n$ commitments. One for each permutation's edge on both left and right side. A commitment allows a verifier to interactively (with the mixer) check the property. On the left i–th commitment $C_L(i)$ should reveal an index n such that $\pi_L(n) = i$. On the right i–th commitment $C_R(i)$ should reveal the value $\pi_R(i)$.

A practical way to implement commitments would be to use a cryptographic hash function h. For every commitment i, the mixer concatenates the commitment's value and some noise $\gamma_i \in \{0,1\}^\star$ used as a salt. Let $C_R(i) = h(\pi_R(i), \gamma_i)$. To verify a commitment, the mixer publishes $\pi_R(i)$ and γ_i, and the verifier can calculate h on its own and compare it with $C_R(i)$. On the left side the procedure is similar.

Verification

After the output and the commitments are published the mixer gets a list of n bits $\langle b_1, b_2, \ldots, b_n \rangle$. Depending on the b_i he opens a commitment for left or right edge. Formally if $b_i = 0$ then the mixer opens $C_L(i)$ and if $b_i = 1$ the mixer opens $C_R(i)$. Once an edge $\langle n, m \rangle$ is revealed, the mixer must present a non–interactive zero–knowledge proof that decrypted n–th item is same as m–th item in the next column. A verification can only be done once.

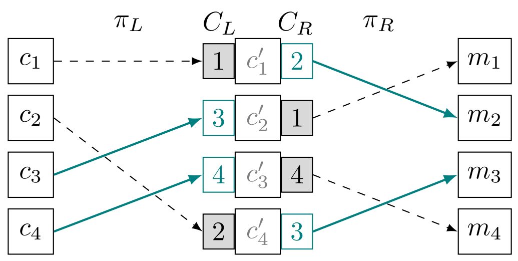

Figure 3: Opening commitments for challenge bits $\langle 1, 0, 0, 1\rangle$. Colored elements are revealed. Gray elements remain secret.

A verifier now has no knowledge of any edge of permutation π. Both π_L and π_R were randomly selected by the server, after choosing to reveal the left side nothing is determined about the right side (since we do not validate the right edge nor its decryption). Unfortunately, any unchecked commitment might be hiding a manipulation. In some cases challenge bits will not be able to detect an error.

Completeness

It is worth pointing out that a properly working decryption mix network is always able to pass the RPC verification. Again, since π_L is indeed a permutation we can commit to its inverse $C_L = \pi_L^{-1}$ and symmetrically on the other side $C_R = \pi_R$. We can also provide a zero–knowledge proof for each decryption since we really have established the plain texts by decrypting them.

Soundness

In case of RPC, the verification can only be done once. Otherwise privacy would be breached. As a result there is no soundness property for RPC, at least when using the standard definition. Instead of asking about a number of rounds, we can ask about a number of errors and associated probability of detecting any of them. As we will show, this probability approaches 1.

1.3 Probability of detecting a manipulation

As Shahram Khazaei et al. noticed, the probability of detecting mix–network's malfunction should depend on a particular attack [7]. For now, suppose there is a single modified entry on the output.

Single modified entry

Lemma 1. *When a single entry was modified, it will be detected with the probability of at least* $\frac{1}{4}$.

Proof. Let us analyse left and right sides separately. WLOG we assume that the mixer was supposed to decrypt the first input as the first output on a substage. Instead he failed and replaced the output with its own text. If the entry was modified on the left, one of the following cases happens:

1. *We committed to $C_L(1)$ for which we cannot prove the correctness of decryption.* It means that when a verifier opens $C_L(1)$ we fail the verification. This happens with the probability of $\frac{1}{2}$.
2. *We committed to $C_L(1)$ for which we can prove the correctness of decryption.* It means, that there exists an input which decrypted yields our message. Notice that since only one entry was modified, we can prove the correctness of decryption for all other entries. No commitment can point to the first input. We committed n times to $n-1$ inputs and for one input we committed at least twice. To detect a double commitment we have to open both of them. This happens with the probability of $\frac{1}{4}$.

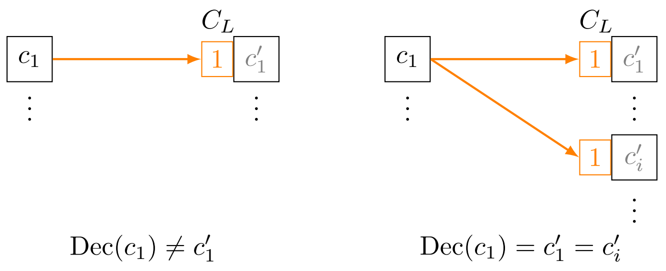
C_L
c_1
1
c'_1
$\mathrm{Dec}(c_1) \neq c'_1$
C_L
c_1
1
c'_1
1
c'_i
$\mathrm{Dec}(c_1) = c'_1 = c'_i$

Figure 4: Both cases on the left visualized.

On the right side the cases are similar:

1. *We committed to $C_R(1)$ for which we cannot prove the correctness of decryption.* It means that when a verifier opens $C_R(1)$ we fail the verification. This happens with the probability of $\frac{1}{2}$.
2. *We committed to $C_R(1)$ for which we can prove the correctness of decryption.* It means, that there exists an output which matches our decrypted first input. Again we committed n times to $n-1$ outputs and for one output we committed at least twice. To detect a double commitment we have to open both of them, which happens with the probability of $\frac{1}{4}$.

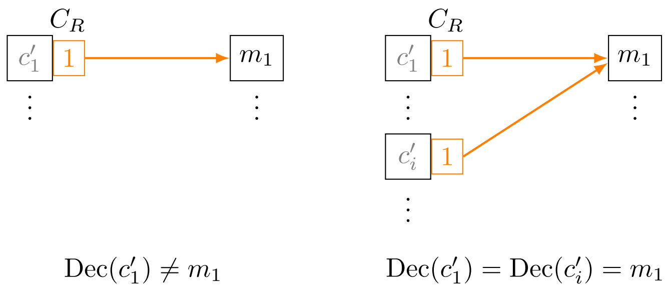
C_R
c'_1
1
m_1
C_R
c'_1
1
m_1
c'_i
1
$\mathrm{Dec}(c'_1) \neq m_1$
$\mathrm{Dec}(c'_1) = \mathrm{Dec}(c'_i) = m_1$

Figure 5: Both cases on the right visualized.

■

Multiple manipulations

Let us consider a mix network with multiple manipulated outputs. We will focus on the left substage only, as the right substage is symmetrical. To calculate the probability of finding any of the errors we will use induction. Let a_i be the probability of detecting any error when checking first i rows. Without checking any rows no error will be detected, thus

$$a_0 = 0$$

Now suppose probability of detecting any error among first i rows is a_i. One of these 3 cases takes place.

1. We link the $(i+1)$–th message to any cryptogram that is not an encrypted message. When revealing $C_L(i+1)$ we will detect the manipulation as the mixer would no be able to present a zero knowledge proof of decryption.
2. We connect the $(i+1)$–th message to a cryptogram that decrypts to our message and the cryptogram has not been connected to any row with same message. As this does not introduce any error it does not alter our probability.
3. Like in the 2. case, but the the cryptogram had already j properly decryptable messages linked.

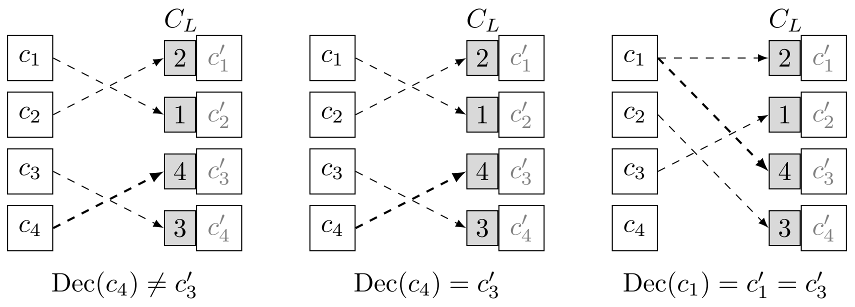
C_L
c_1
c_2
c_3
c_4
2
1
4
3
c'_1
c'_2
c'_3
c'_4
$\mathrm{Dec}(c_4) \neq c'_3$
$\mathrm{Dec}(c_4) = c'_3$
$\mathrm{Dec}(c_1) = c'_1 = c'_3$

Figure 6: All three cases visualized for 3. step – a_3 is calculated.

We can use these cases to make a recurrence relation:

$$a_{i+1} = \begin{cases} \frac{1}{2}a_i + \frac{1}{2}, & \text{in the 1. case} \\ a_i, & \text{in the 2. case} \\ a_i + (1 - a_i)\frac{j}{2(j+1)}, & \text{in the 3. case} \end{cases}$$

In both 1. and 3. case we use the law of total probability. In the first case the event is checking $(i + 1)$-th row and its complement. In the second finding an error among first i rows and its complement.

Denote k as the total number of manipulations. Observe that it is equivalent to number of times 1. case or 3. case occurred.

Lemma 2. *The probability of detecting any of k manipulations is*

$$a_n \geq 1 - \left(\frac{3}{4}\right)^k$$

Proof. Observe that relations in both (1. and 3.) cases are linear functions of a_i. For any $a_i \in [0, 1]$ the value in case 1 will not exceed $\frac{1}{4} + \frac{3}{4}a_i$ and for any $a_i \in [0, 1]$ and any j:

$$a_i + (1 - a_i)\frac{j}{2(j+1)} \geq a_i + (1 - a_i)\frac{1}{2(1+1)} = \frac{1}{4} + \frac{3}{4}a_i$$

Calculating a_{i+1} when 1. case or 3. case takes place (which is k times):

$$a_{i+1} \geq \frac{1}{4} + \frac{3}{4}a_i \quad \Longrightarrow \quad a_n \geq 1 - \left(\frac{3}{4}\right)^k$$

■

2 Improving detection of errors in RPC

Although detecting an error with the probability of $\frac{1}{4}$ is quite high, it can be improved. Since what a verifier is struggling with is finding a witness for π

not being a permutation, we can help him by forcing a mixer to publish more commitments. A permutation is an endomorphic bijection by definition, which means we can ask the mixer to commit to both an edge and its inverse without revealing any more information.

Formally, on the left, the i–th commitment $C_L(i)$ still holds index j such that $\pi_L(j) = i$. Additionally, $\overline{C_L}(j)$ holds $\pi_L(j)$. On the right side symmetrically $C_R(i)$ does not change and $\overline{C_R}(i)$ holds an index j such that $\pi_R(j) = i$. In total we published $4n$ commitments.

In verification part of the protocol the verifier still provides n challenge bits $\langle b_1, b_2, \ldots, b_n \rangle$. For each bit the mixer opens either left or right commitment. The verifier checks if $j = C_L(i)$ and $i = \overline{C_L}(j)$ (and similarly on the right side). As before the mixer shows a non–interactive zero–knowledge proof that decryption was correct.

Observe that commitments with bars are stronger than their barless counterparts. The mixer is forced to make a bar–commitment for every predecessor j and every successor i. If any of k successors is not committed, we can detect it by asking to verify it with probability of $\frac{1}{2}$ for each successor. Moreover, by definition of a commitment (as a function) it must commit to every predecessor. Thus, for given i by asking to reveal such j and a commitment $\overline{C_L}(j) = i$ we are already convinced that $C_L(i) = j$ and can skip publishing and verifying barless commitments.

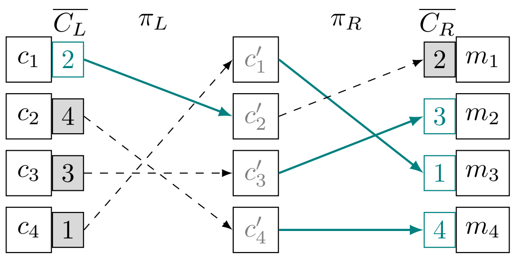

Figure 7: Opening commitments for challenge bits $\langle 0, 1, 0, 0 \rangle$. Colored elements are revealed. Gray elements remain secret.

The final protocol would look like this:

Protocol 3 Fixed RPC

1. The mixer establishes keys pk and sk, then he waits for all n inputs $\langle c_1, c_2, \ldots, c_n \rangle$ to be delivered.
2. The mixer chooses π_L and π_R, and decrypts the inputs twice: when shuffling on the left and when shuffling on the right.

$$\langle c'_1, \ldots, c'_n \rangle = \left\langle \mathrm{Dec}_{sk}\left(c_{\pi_L(1)}\right), \mathrm{Dec}_{sk}\left(c_{\pi_L(2)}\right), \ldots, \mathrm{Dec}_{sk}\left(c_{\pi_L(n)}\right) \right\rangle$$

$$\langle m_1, \ldots, m_n \rangle = \left\langle \mathrm{Dec}_{sk}\left(c'_{\pi_R(1)}\right), \mathrm{Dec}_{sk}\left(c'_{\pi_R(2)}\right), \ldots, \mathrm{Dec}_{sk}\left(c'_{\pi_R(n)}\right) \right\rangle$$

3. The mixer publishes all n outputs $\langle m_1, m_2, \ldots, m_n \rangle$ and $2n$ commitments: $\overline{C_L} = \pi_L$ and $\overline{C_R} = \pi_R^{-1}$.
4. The verifier gives n challenge bits $\langle b_1, b_2, \ldots, b_n \rangle$.
5. The mixer reveals both j and $\overline{C_L}(j) = i$ iff. $b_i = 0$ and both j and $\overline{C_R}(j) = i$ iff. $b_i = 1$. He also provides non–interactive zero–knowledge proofs on the correctness of decryption on revealed edges.
6. The verifier validates data provided by the mixer.

Single error

Before looking at the general case, let us focus on single errors first.

Lemma 3. *The probability of detecting a single error on the left is* $\frac{1}{2}$

Proof. Again (like in analysis of Jakobsson's RPC) suppose a single error occurs on the left side. WLOG let an error to occur in the first row. If $b_1 = 1$ (meaning we do not check the edge on the left), then no error will be detected (as the rest of the rows are errorfree). When $b_1 = 0$ one of the following cases takes place:

1. *There is no index k such that $\overline{C_L}(k) = 1$.* As a result we fail a verification.
2. *$\overline{C_L}(k) = 1$ for some index k and decryption at the edge $\langle k, 1 \rangle$ is incorrect.* Again in this case we fail the verification as well.

When $b_1 = 0$ the mixer fails the verification. ■

The right side is symmetrical and has the same probability of detecting a single error.

Multiple errors

Lemma 4. *If a single server manipulates k entries then* FixedRPC *protocol will detect it with probability* $1 - 2^{-k}$.

Proof. For this analysis let k be the number of errors, the goal is to find what is the smallest possible probability of detecting any of k errors. Again we will use induction. Denote a_i as the probability of detecting an error by verifying only first i rows. Without verifying anything we have no proof of an error, therefore,

$$a_0 = 0$$

When trying to calculate a_{i+1} one of the cases takes place:

1. No commitment has been published that links to $(i+1)$–th message. In other words: for all j $\overline{C_L}(j) \neq i+1$ when verifying the left side or for all j $\overline{C_R}(j) \neq i+1$ when on the right side.
2. At least one commitment has been published that links to $(i+1)$–th message and the decryption on any of the edges was correct.
3. At least one commitment has been published that links to $(i+1)$–th message and the decryption on all edges was incorrect.

When choosing to verify a row, where the 1. case or the 3. case happened the verification will always fail, and we can choose to verify this row with the probability of a half. When at the $(i+1)$–th row the 2. case happens we will pass the verification regardless of the choice to verify the row or not. These cases render a recurrence relation:

$$a_{i+1} = \begin{cases} \frac{1}{2}a_i + \frac{1}{2}, & \text{in the 1. case and the 3. case} \\ a_i, & \text{in the 2. case.} \end{cases}$$

and a solution

$$a_n = 1 - \left(\frac{1}{2}\right)^k$$

■

Why does the improvement matters?

It is not unheard of for some election results to be close. There were several elections where the winner had only a few votes more that his best opponent. In such cases the mixer only has to alter very little votes to change the winner and he might be encouraged to cheat.

Number of changed votes	Prob. of error detection in original RPC[2]	Prob. of error detection in improved RPC
1	25.0%	50.0%
2	43.8%	75.0%
3	57.8%	87.5%
4	68.4%	93.8%
5	76.3%	96.9%
6	82.2%	98.4%
7	86.7%	99.2%
8	90.0%	99.6%
9	92.5%	99.8%
10	94.4%	99.9%
23	99.9%	1-1.2e-7
50	1-5.7e-7	1-8.9e-16
121	1-7.6e-16	1-3.8e-37

Figure 8: Comparison between original and improved RPC regarding the probability of detecting any error.

Observe that for a given probability of detecting any error, the original RPC can change at least twice as many votes as the improved version.

This proposed version of RPC is a ,,drop in replacement" of original RPC and the voting schemes using RPC should consider switching to this improved version as it offers better detection of errors.

verte

Acknowledgements

I would like to express my thanks and gratitude to everybody who had an impact on my education. You helped me discover my interests and deepen my understanding of them. Thank You!

Bibliography

[1] Craig Burton et al. "Using Prêt à Voter in Victoria State Elections." In: *EVT/WOTE* 2 (2012).

[2] David Chaum, Peter YA Ryan, and Steve Schneider. "A practical voter-verifiable election scheme". In: *European Symposium on Research in Computer Security*. Springer. 2005, pp. 118–139.

[3] David Chaum et al. "Scantegrity: End-to-end voter-verifiable optical-scan voting". In: *IEEE Security & Privacy* 6.3 (2008), pp. 40–46.

[4] David L. Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Commun. ACM* 24.2 (Feb. 1981), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/358549.358563. URL: https://doi.org/10.1145/358549.358563.

[5] Michael R Clarkson, Stephen Chong, and Andrew C Myers. "Civitas: Toward a secure voting system". In: *2008 IEEE Symposium on Security and Privacy (S&P 2008)*. IEEE. 2008, pp. 354–368.

[6] Markus Jakobsson, Ari Juels, and Ronald Rivest. "Making Mix Nets Robust For Electronic Voting By Randomized Partial Checking". In: (Mar. 2002).

[7] Shahram Khazaei and Douglas Wikström. "Randomized Partial Checking Revisited". In: (2013). Ed. by Ed Dawson, pp. 115–128.

[8] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. "Formal analysis of chaumian mix nets with randomized partial checking". In: *2014 IEEE Symposium on Security and Privacy*. IEEE. 2014, pp. 343–358.