

Sam Goodwin

A Thousand Variations of Prêt à Voter

A discussion and comparison of the cryptographic protocols involved in some variations of Prêt à Voter

Master's thesis in Mathematical Sciences

Supervisor: Kristian Gjøsteen

June 2019

Sam Goodwin

A Thousand Variations of Prêt à Voter

A discussion and comparison of the cryptographic protocols involved in some variations of Prêt à Voter

Master's thesis in Mathematical Sciences
Supervisor: Kristian Gjøsteen
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Abstract

This thesis is about the cryptographic voting protocol *Prêt à Voter*. This is a protocol that has been used in elections and could be applied to more in the future, whether it be electronic or traditional elections. Some variations that have been or could be attempted are described, and the differences are discussed. Then two particular variations are described in detail. Finally the security of the protocols is demonstrated, in terms of privacy, coercion-resistance, and verifiability.

Contents

I	Introduction	3
1	The <i>Prêt à Voter Classic</i> Variation	3
1.1	The Voter's Experience	4
1.2	Other processes	4
2	The <i>Scratch & Vote</i> Variation	5
2.1	The Voter's Experience	5
2.2	Other processes	6
3	The <i>vVote</i> Variation	6
3.1	The Voter's Experience	6
3.2	Other processes	7
II	Tools	8
4	Paillier Cryptosystem	8
4.1	Outline of the Paillier cryptosystem	8
4.2	Useful Properties	10
5	Definitions	11
6	Protocols	12
6.1	Protocol for n -th powers	12
6.2	Protocol for 1-out-of-2	14
6.3	Cut-and-Choose Protocol	15
6.4	Mixnet Shuffle	15
7	Fiat-Shamir Heuristic	16
7.1	Random Oracle Model	16
7.2	Application to P_n	17

III	Choices for subprotocols of two <i>Prêt à Voter</i> Variations	18
8	Implementation of <i>Paillier Prêt à Voter</i> and <i>Prêt à Voter with Mixnets</i>	18
8.1	Ballot Generation	18
8.2	Verifying the correctness of the ballots	19
8.3	Vote Casting	20
8.4	Vote Processing and Counting	20
8.5	Auditing	21
9	Player Interactions in <i>Paillier Prêt à Voter</i> and <i>Prêt à Voter with Mixnets</i>	22
9.1	Players	23
9.2	Description of the Processes	24
IV	Security Proofs for <i>Paillier Prêt à Voter</i> and <i>Prêt à Voter with Mixnets</i>	27
10	Security Goals	27
11	Security Proofs for Privacy	29
11.1	<i>Paillier Prêt à Voter</i>	29
11.1.1	Privacy Game	31
11.1.2	Security Proof	31
11.2	<i>Prêt à Voter with Mixnet</i>	36
11.2.1	Privacy Game	38
11.2.2	Security Proof	38
12	Proof of Coercion-Resistance	40
12.1	<i>Paillier</i>	41
12.2	<i>Mixnet</i>	42
13	Verification Proof	42
13.1	<i>Paillier</i>	42
13.2	<i>Mixnet</i>	43
14	Remaining Attacks	43
15	Concluding Remarks	44

Part I

Introduction

Democracy is one of the most important and valued aspects of our modern society, and elections are the way in which we apply it. It is crucial that we continue to work on improving our current practices worldwide to ensure voters have security and privacy. Cryptography is often used for this purpose, and in this paper we will discuss a recent suggestion for a secure voting protocol: *Prêt à Voter*.

One important factor this protocol attempts to address, that other systems do not, is complete privacy for the voter such that no one can find out how an individual voted, even with access to their submitted ballot paper. This avoids the problem of corrupt institutions, governments or coercers who may threaten the voter's liberty to vote for who they want.

Electronic voting, which has both positive and negative aspects, is another topic to consider but the underlying protocols are important to address, whatever method of voting is used. *Prêt à Voter* is such a protocol, with many variations and applications, whether in electronic or traditional elections with physical ballot papers.

We will describe a few of these variations in Part I, and discuss the differences between them. We will then outline some definitions, protocols and concepts to be applied throughout the paper in Part II. In Part III, we will outline two specific variations and look at these in more detail, discussing the choices for subprotocols and describing the specific players and processes. Finally in Part IV, we will define our security goals and prove that our variations are secure and according to our definition, before demonstrating some possible attacks that could be discussed in future papers.

First, we will describe the basic elements of some variations that have been proposed and the differences between them, specifically the *Prêt à Voter Classic*, *Scratch & Vote* and the *vVote* variations.

1 The *Prêt à Voter Classic* Variation

The original version of *Prêt à Voter* was created by Peter Ryan in 2005. It is a cryptographic protocol that aims to guarantee privacy in election voting systems. In this section I will go through the processes involved in voting in *Prêt à Voter Classic*, describing the players and their roles, and then describe the processes taking place in the rest of the protocol. The other variations discussed below are all based on this protocol, so the basic principles remain the same.

Candidate List	Your Vote
Karl	
Rosa	
Thomas	
Helen	X
	F8A2kL
<i>Destroy this side</i>	<i>Keep this Side</i>

Table 1: An example of a *Prêt à Voter* ballot.

1.1 The Voter’s Experience

We call our registered voter Alice. Alice chooses one ballot, as shown in Table 1, from a pile of sealed envelopes at random, and takes it to a private voting booth. In the private booth, Alice makes her vote by putting a cross on the right hand side, against her chosen candidate. The order of the candidates is randomised for each ballot so it is crucial for privacy that the booth is private and no one knows the order of her ballot except Alice herself.

Then, Alice detaches and discards the left hand side (with the order of candidates on), and keeps the encrypted right hand side. Now Alice leaves the booth, and may cast this right hand side as her vote, in the presence of an official. The ballot is read by a device which records the cryptographic information at the bottom; we can call this the ballot *cipher*, or ψ . We use τ as an indicator for which cell Alice crossed. A digital signature is then computed for $\{\tau, \psi\}$, and this is printed onto the receipt, which she may keep.

The device that stores the votes is often a Web Bulletin Board (WBB). This is an authenticated public broadcast channel which publicly displays certain information such as the submitted ballots, and proofs of correct decryption, so that Alice may verify her vote was cast as intended, and public auditors may verify the protocols were carried out correctly. Other players may include Helper Organisations, which may assist in receipt verification or with the voting process in general. It is preferable that they are independent from the Officials. The only essential players are Alice, the Officials, and the WBB (or equivalent) which stores and displays the encrypted votes and decrypts them for tallying. [1]

1.2 Other processes

The first stage of any *Prêt à Voter* system is the Ballot Generation. The Ballot consists of two halves which can be detached from each other after the voting process. On the left hand side, LHS, is a detachable list of Candidates, placed in random order for each ballot. On the RHS are the boxes the voter can choose from, and some encrypted information, so that the system can later decipher the candidate order.

Once created, each individual randomised ballot must be sealed in an envelope to ensure only Alice knows the order of the candidates on her ballot.



Candidate List	Your Vote
Karl	
Rosa	
Thomas	
Helen	X
	
	

Table 2: An example of a *Scratch & Vote* ballot.

An alternative method is for the randomised ballots to be printed as needed, within the privacy of the booth, but this relies on an extra player, the printer, not being manipulated or observed by outside sources. Other variations do use this method of printing on demand.

After the vote has been cast, there is the Vote Processing stage, which includes mixing, decrypting and tallying. The specifics of this stage vary depending on the type of *Prêt à Voter* being used, however the crucial idea is that no one can perform end-to-end matching.

The final stage is Auditing, in which we ensure end-to-end verifiability. Individual voters are able to verify their votes were cast correctly, and public auditors can check that the mixing, decrypting and tallying were performed correctly, and the system is working as intended.[2]

2 The *Scratch & Vote* Variation

The *Scratch & Vote* variation was invented by Ben Adida in 2006. It takes the concept of *Prêt à Voter* paper voting, but encodes the candidate order in a more novel way, utilising barcodes and scratch surfaces.

2.1 The Voter’s Experience

The first stage of the *Scratch & Vote* system is similar. Alice obtains a ballot with randomised candidate order on the left, which the officials should not see, and scannable boxes on the right. As in *Prêt à Voter*, these two halves can be detached. In this variation a 2D-barcode is embedded below the boxes, and a scratch surface below that. The scratch surface should also be separable from the right half. We demonstrate what this ballot looks like in Table 2.[3]

Once the vote has been made, the halves are separated and the left half is discarded. As long as the scratch surface is intact, the Official detaches it and discards it. The checkmark and barcode can then be scanned, and kept as a receipt. As with *Prêt à Voter Classic*, the votes are verifiable by individual voters and the processes are verifiable by public auditors.

2.2 Other processes

An advantage of the 2D-barcode and scratch surface is that the auditing can be done immediately and without election official intervention. The ciphertexts required for the candidate order on one ballot are encrypted in the 2D-barcode, whilst the randomisation values used to generate these ciphertexts are printed under the scratch surface.

Before voting, the voter may scratch a random spare ballot, in order to verify that the candidate order is as the randomisation value suggests. This essentially is a way of verifying the correctness of the ballots ‘live’, so that the voters feel more secure voting. Then they destroy this ballot as scratching the surface compromises the secrecy, and they take a fresh ballot to vote with.

In this system a Help Organisation of some format would be useful in explaining the system fully to ensure voters do not vote with a spoiled ballot, and that they may verify the correctness of the ballots by spoiling a spare one and ensuring the randomisation values correspond to the encrypted candidate order. The Officials also play an important role, in discarding the scratch surface part of the ballot, unscratched, with plenty of observers, including the voter.

3 The *vVote* Variation

The *vVote* system is an electronic voting scheme based on the *Prêt à Voter* system, which was actually used in the Victorian State Election in 2014. [4] As such, it is specialised for the Australian voting system. It is a good example of how *Prêt à Voter* protocols may be adapted for specific situations, and shows they are considered secure enough to be used in large elections.

3.1 The Voter’s Experience

First, Alice is marked off the electoral roll and her ballot is printed with the candidates in a random order, along with a QR code encrypting the candidate order. At this stage, Alice may ask for the ballot to be audited to verify it matches the encryption the system is committed to. In this case, a new Candidate List will be printed afterwards as an audited one cannot be used to vote. Once Alice has verified the system is working, she takes her new Candidate List into a private booth.

In the voting booth, an Electronic Ballot Marker, commonly a tablet, is used by Alice to scan the QR code, which launches the vote capture application, on which Alice can enter her vote. In the Australian system, candidates are ordered in preference, so this list is then printed as a Preferences Receipt. This receipt doesn’t list the actual candidates, instead just the order, for example {4, 1, 2, 5, 3} if there are 5 candidates, corresponding to Alice’s randomised candidate order. Then, Alice can verify that the preferences match the candidates on the ballot she intended to vote for, and if it is incorrect, Alice may cancel the vote at this point by supplying the Candidate List to an official. Once Alice is content with her Preference Receipt, the Candidate List must be destroyed for

secrecy. Alice may keep Preference Receipt, and can use it later for verification by checking the WBB to ensure her vote was recorded.

3.2 Other processes

The system has many key components. First, the print-on-demand printer is a computer and printer which generates randomised candidate orders along with the QR code encrypting this order, and then prints them on demand. This works in conjunction with some kind of Randomness Generation Service (RGS) which provides the random element for the process. This ensures no one has access to what any individual voters' ballot will look like.

The Electronic Ballot Marker (EBM) is a computer that enables the voter to fill in their ballot and sends the results to the WBB. This variation also sends the votes through a series of Mixnet servers, in order to shuffle the votes and ensure votes cannot be linked to individual voters, so that privacy is guaranteed. We will discuss Mixnet shuffles in more detail for one of our variations that we prove to be secure.

Part II

Tools

In this section we will outline some tools which will be used or referred to throughout the rest of the paper, as we describe our particular variations of *Prêt à Voter* and prove they are secure. Specifically, we will outline how Paillier Encryption, Cut-and-Choose and Mixnets work, define some properties related to the security of cryptosystems and present some protocols which will be used in the security proof.

4 Paillier Cryptosystem

In this section we will outline the Paillier cryptosystem, and describe some of its properties and why they are useful when applying it to the *Prêt à Voter* protocol.

Pascal Paillier invented the public key Paillier cryptosystem in 1999 [5], and it is a probabilistic asymmetric algorithm. It is considered to be applicable to various situations such as online voting, computation outsourcing and electronic cash. It is also useful in the *Prêt à Voter* protocol due to some of its properties that we will discuss soon.

4.1 Outline of the Paillier cryptosystem

We will now give a technical outline of the algorithms involved in key generation, encryption and decryption within the Paillier cryptosystem. Note that in these schemes the notation $\frac{x}{y}$ denotes finding the quotient, i.e. the largest integer $z \geq 0$ s.t. $x \geq zy$.

Key Generation

1. Choose two large, distinct primes p, q at random such that $\gcd(pq, (p-1)(q-1)) = 1$.
2. Compute $n = pq$ and $\lambda = \text{lcm}(p-1, q-1)$.
3. Select a random integer $g \in \mathbb{Z}_{n^2}^*$
4. Check that n divides the order of g by ensuring $\mu = \frac{((g^\lambda \bmod n^2)^{-1}) - 1}{n}$ exists.

Now we have obtained the public encryption key, which is (n, g) , and the private decryption key, (λ, μ) .

Encryption Assume we have a message m , to encrypt such that $0 \leq m < n$.

1. Select a random r such that $\gcd(r, n) = 1$.
2. Compute the ciphertext with: $c = g^m \cdot r^n \bmod n^2$.

Decryption

1. Take as input the ciphertext c .
2. To decrypt compute m as $m = \frac{c^\lambda \bmod n^2 - 1}{n} \cdot \mu \bmod n$.

So this summarises the Paillier Encryption system. It is worth noting that one way of ensuring that $\gcd(pq, (p-1)(q-1)) = 1$ is making sure p, q are of equal length. In this case, there is a simpler variant of the key generation, described in ‘Introduction to Modern Cryptography: Principles and Protocols’ [6] that we can quickly outline. In this variant we set $g = n + 1$, $\lambda = \varphi n$ and $\mu = \varphi(n)^{-1} \bmod n$.

We will stick with the more general variant for our purposes. Also, there is a further generalization of the cryptosystem, known as the ‘Dåmgard-Jurik cryptosystem’, which uses computations modulo n^{s+1} instead of n^2 . Since our list of messages needing to be encrypted (the list of candidates) is unlikely to be very large in practice, we don’t need this generalization so we will use the original.

We will prove the correctness of this cryptosystem to show it decrypts correctly. First we must define the n -th residuosity class of w

Definition 4.1. The n -th residuosity class of w with respect to g is the unique integer $x \in \mathbb{Z}_n$ such that $\exists y \in \mathbb{Z}_n^*$ such that

$$w = g^x \cdot y^n \bmod n^2$$

and is denoted $\llbracket w \rrbracket_g$.

Theorem 4.2. *The Paillier cryptosystem decrypts correctly.*

Proof. We must show that

$$m = \frac{c^\lambda \bmod n^2 - 1}{n} \cdot \mu \bmod n = \frac{c^\lambda \bmod n^2 - 1}{n} \cdot \frac{((g^\lambda \bmod n^2)^{-1}) - 1}{n} \bmod n$$

First we define the L-function as $L(u) = \frac{u-1}{n}$ for u s.t. $\{un^2 | u = 1 \bmod n\}$. Then we notice that this can be rewritten as

$$m = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n$$

It can be shown that for any $w \in \mathbb{Z}_{n^2}^*$ and $g_1, g_2 \in \mathbb{Z}_{n^2}^*$, the identity holds such that

$$\llbracket w \rrbracket_{g_1} = \llbracket w \rrbracket_{g_2} \llbracket g_2 \rrbracket_{g_1} \bmod n.$$

Using this identity we can apply it to our equation above to get

$$\frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} = \frac{\lambda \llbracket c \rrbracket_{1+n}}{\lambda \llbracket g \rrbracket_{1+n}} = \frac{\llbracket c \rrbracket_{1+n}}{\llbracket g \rrbracket_{1+n}} = \llbracket c \rrbracket_g \bmod n$$

From the definition we have $\llbracket c \rrbracket_g \bmod n$ is such that $c = g^{\llbracket c \rrbracket_g} \cdot r^n \bmod n^2$, hence $\llbracket c \rrbracket_g = m \bmod n$ as required. \square

4.2 Useful Properties

There are two properties of Paillier Encryption we will discuss that are particularly relevant and helpful in creating a secure voting protocol. These are its non-deterministic encryption and its homomorphic properties.

Non-deterministic Encryption The presence of the randomly chosen integer r in the encryption process is particularly useful in creating a secure cryptosystem. This is because it creates a non-deterministic encryption, i.e. for a given message and key, the system does not create the same ciphertext each time.

This makes it much more difficult for an adversary to see repeated ciphertexts or patterns and attempt to link them to messages (in our case candidates). This means the voters have more privacy than if all voters who voted for a candidate used exactly the same ciphertext.

Homomorphic Properties The encryption function for Paillier is additively homomorphic which gives it useful features for the process of tabulating the final votes without compromising voter privacy. We will go more into depth on this later, but essentially this property allows the messages to be added together without losing information, assuming the messages are chosen correctly.

Also multiple ciphertexts may be decrypted together as the product of two ciphertexts will decrypt to the sum of their plaintexts, i.e.

$$D(E(m_1, r_1) \cdot E(m_2, r_2) \bmod n^2) = m_1 + m_2 \bmod n$$

We will now show that Paillier Encryption has the homomorphic property over addition.

Theorem 4.3. *Paillier Encryption is additively homomorphic*

Proof.

$$\begin{aligned} E(m_1, r_1) \cdot E(m_2, r_2) \bmod n^2 &= (g^{m_1} r_1^n \bmod n^2) \cdot (g^{m_2} r_2^n \bmod n^2) \\ &= g^{m_1+m_2} (r_1 \cdot r_2)^n \bmod n^2 \\ &= E(m_1 + m_2) \end{aligned} \tag{1}$$

So this shows it has the homomorphic property over addition, and trivially, applying the correct decryption to both sides we have also shown that multiple ciphertexts can be decrypted together. \square

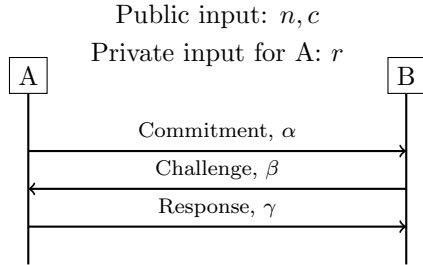


Figure 1: Σ -Protocol

5 Definitions

We now define some useful properties that will be referred to later, specifically properties of proofs for use in showing that our cryptosystem is secure according to our security goals. For the definitions of properties of a proof, we set up the conversation that the proof takes place in. It is between a prover, P, and a verifier, V. P claims something and they must prove to V that this is true, who will then accept or reject the proof.

Definition 5.1. A proof has **Completeness** if, assuming P's claim is correct and that the verifier V is honest, V will accept the proof every time, i.e. with probability 1.

Definition 5.2. A **Σ -Protocol** is a protocol in the form shown in Figure 1. It is a conversation between A and B of the form (α, β, γ) . There is a public input which both receive, (n, c) , and a private input for A, witness r . The public input is tied to the private input in some way. The aim of the protocol is for A to prove to B that they know the witness r without revealing it to B. A sends a commitment α , receives a challenge β , then sends the response γ .

This format is used for A to prove to B, for example, that c encrypts m , without revealing the witness w used in this encryption. We will use it later in our protocols and apply Paillier encryption to a Σ -Protocol to prove correct decryption.

Definition 5.3. A proof has **Special Soundness** if, given two accepting conversations from Σ -Protocols, (α, β, γ) and $(\alpha, \tilde{\beta}, \tilde{\gamma})$ with the same input x and $\beta \neq \tilde{\beta}$, it is easy to compute the witness r .

Definition 5.4. A **Zero-Knowledge** proof is such that, if P's claim is true then there is no V, even if it is cheating, that can learn anything except this fact from the protocol. For a non-interactive protocol, Zero-Knowledge is known as **non-interactive ZK** or **NIZK**.

Definition 5.5. A **Special Honest Verifier Zero Knowledge** proof, **SHVZK**, is such that there exists a simulator S which with input c, β outputs an accepting

conversation (α, β, γ) , with the same probability distribution as conversations between (honest) P and V with input c and challenge β .

Note that we will use the simulator S in a protocol later.

6 Protocols

We now outline some protocols which will be used later to represent a simulation of the *Prêt à Voter* Protocol, in order to prove security properties.

6.1 Protocol for n -th powers

This protocol between the prover, P, and the verifier, V, has the purpose of P being able to prove to V that they have the private value r such that $c_0 = r^n$. This also proves to the verifier that c_0 is indeed an n 'th power.

We will use this protocol later in the security proof of *Prêt à Voter* to show that it correctly decrypts the ciphertexts.

Input: n, c_0

Private Input for P: r such that $c_0 = r^n$

1. P chooses ρ , sends $\alpha \leftarrow \rho^n$ to V.
2. V chooses a random challenge, β , and sends it to P.
3. P calculates $\gamma \leftarrow \rho r^\beta \bmod n$ and sends to V.
V verifies that $\gamma^n = \alpha c_0^\beta \bmod n^2$.

If the verification is a success, and c_0, α, γ are prime to n , it shows that c_0 is an n 'th power. We refer to this protocol as P_n from now on.

Proof of Correct Decryption We examine the case where the voter would like to verify their vote was correctly decrypted, without knowing or leaking any additional information about the protocol, using an HVZK proof, (Honest Verifier Zero-Knowledge). [9] Note that this is not a zero-knowledge proof, and relies on an honest verifier, however we can obtain security in the random oracle model as we will be using a non-interactive variant based on the Fiat-Shamir heuristic.

Suppose prover P presents a voter V with ciphertext c , claiming it encodes m . To prove this is equivalent to proving that $c_0 = c(1+n)^{-m} \bmod n^2$ is an n 'th power, by a simple rearrangement of the encryption formula. In other words, we must show there exists r such that $r^n = c_0$. The *Protocol for n -th powers*, P_n is a Protocol which fulfills this purpose, and we now prove it is a Σ -Protocol with Completeness, Special Soundness and SHVZK, and hence it decrypts correctly.

Theorem 6.1. P_n is a Σ -Protocol with Completeness, Special Soundness and SHVZK.

Proposition 6.2. P_n is a Σ -Protocol.

Proof. It is trivial from comparing their forms that the format of the conversation P_n is the same as that of a Σ -Protocol. \square

Proposition 6.3. P_n has Completeness.

Proof. From the definition, we require that if P and V are honest, then V should accept the proof with overwhelming probability. For V to accept, we just require that $\gamma^n \equiv \alpha c_0^\beta \pmod{n^2}$. So assuming P has ρ and is honest, this is clear, as we have:

$$\gamma^n \equiv (\rho r^\beta)^n \equiv \rho^n r^{n\beta} \equiv \alpha c_0^\beta \pmod{n^2}$$

So the protocol has completeness. \square

Proposition 6.4. P_n has Special Soundness.

Proof. Suppose (α, β, γ) and $(\alpha, \tilde{\beta}, \tilde{\gamma})$ are obtained from the P_n protocol, $\beta \neq \tilde{\beta}$. This gives us

$$\begin{aligned}\gamma^n &\equiv \alpha x^\beta \pmod{n^2} \\ \tilde{\gamma}^n &\equiv \alpha x^{\tilde{\beta}} \pmod{n^2}\end{aligned}$$

Which leads us to:

$$\left(\frac{\gamma}{\tilde{\gamma}} \pmod{n}\right)^n = c_0^{\beta - \tilde{\beta}} \pmod{n^2}$$

We have that $\gcd((\beta - \tilde{\beta}), n) = 1$ and so there must exist a, b such that $an + b(\beta - \tilde{\beta}) = 1$. Recall that only P knows r such that $c_0 = r^n \pmod{n^2}$.

Let $c'_0 = c_0 \pmod{n}$ and $\tilde{r} = c'_0{}^a \left(\frac{\gamma}{\tilde{\gamma}}\right)^b \pmod{n}$.

We want to show that $\tilde{r} = r$, as this shows that the protocol has special soundness. Note that:

$$c'_0{}^n = (c_0 \pmod{n})^n = c_0^n$$

Then, we can calculate

$$\tilde{r}^n = (c'_0{}^a \frac{\gamma^b}{\tilde{\gamma}^b})^n = (c'_0{}^a)^n \left(\frac{\gamma}{\tilde{\gamma}}\right)^{bn} = c_0^{an} c_0^{b(\beta - \tilde{\beta})}$$

Since $an + b(\beta - \tilde{\beta}) = 1$ this gives us

$$c_0^{an} c_0^{b(\beta - \tilde{\beta})} = c_0 \pmod{n^2}$$

Hence, we have $\tilde{r}^n = c_0 \pmod{n^2}$, so $\tilde{r} = r$ as required, and the protocol has Special Soundness. \square

Proposition 6.5. P_n is SHVZK.

Proof. Finally we need to prove the protocol is Special Honest Verifier Zero Knowledge, such that there exists an adversary A with input $\{c_0, \beta\}$ that can successfully output an accepting conversation $\{\alpha, \beta, \gamma\}$ with the same probability distribution as between an honest P and V, with input c_0 and V's challenge being β . So we let the input for S be $c_0 \in \mathbb{Z}_{n^2}^*, n$ and β . We choose a random $\gamma \in \mathbb{Z}_n^*$ and let

$$\alpha = \gamma^n c_0^{-\beta} \bmod n^2$$

This gets us an accepting conversation as required.

In the case of a conversation between P and V, P chooses ρ uniformly at random and computes α, γ . In this case, S chooses random γ in the same manner, then computes α . Furthermore, both are chosen from the group modulo (and relatively prime to) n . Therefore S will have the same probability distribution as the real conversation between P and V, so P_n has the property SHVZK. \square

So we have proven our proposition, hence completing our proof of correct decryption.

6.2 Protocol for 1-out-of-2

This Protocol is used for P to prove to V that a ciphertext c is either the encryption of m_0 or m_1 .

<p>Input: $c_b, c_{1-b} \in \mathbb{Z}_{n^2}^*, n$ Private Input for P: $r \in \mathbb{Z}_n^*$ such that $c_b = r^n$</p> <ol style="list-style-type: none"> 1. P chooses $\rho_b \in \mathbb{Z}_n^*$ at random and a random challenge β_{1-b}. Then P applies the S simulation described in the definition of SHVZK, with input n, c_{1-b}, β_{1-b}. From S, P receives the conversation $(\alpha_{1-b}, \beta_{1-b}, \gamma_{1-b})$. P sends $\alpha_b \leftarrow \rho_b^{n^2}$ and α_{1-b} to V. 2. V chooses a random challenge $\tilde{\beta}$ and sends it to P. 3. P computes $\beta_b = \tilde{\beta} - \beta_{1-b}$. P computes $\gamma_b = \rho_b r^{\beta_b} \bmod n$. P sends $\beta_b, \beta_{1-b}, \gamma_b, \gamma_{1-b}$ to V. V verifies that: $\tilde{\beta} = \beta_b + \beta_{1-b}$, $\gamma_b^n \equiv \alpha_b c_b^{\beta_b} \bmod n^2$ and $\gamma_{1-b}^n \equiv \alpha_{1-b} c_{1-b}^{\beta_{1-b}} \bmod n^2$.

If the verification is a success, and $\alpha_0, \alpha_1, \gamma_0, \gamma_1$ are relatively prime to n , then this proves that either c_0 or c_1 is the correct encryption, without revealing which. This protocol can be used as a method for verifying the ballots, however for our variation we will not be using it.

6.3 Cut-and-Choose Protocol

The Cut-and-Choose protocol is used for P to prove to V that, for example, n ciphertexts are generated correctly with a very high probability. We assume that P has the randomness, r_i to decrypt the corresponding ciphertexts, c_i such that, if encrypted correctly, $c_i \leftarrow \mathcal{E}(m_i, r_i)$. It goes as follows:

Input: c_i for $i = 1, \dots, n$.

Private Input for P: r_i s.t. $\mathcal{D}(c_i, r_i) = m_i$.

1. V chooses x ciphertexts at random, $0 < x \leq \frac{n}{2}$
2. P reveals r_i and m_i for all x ciphertexts.
3. V verifies that $c_i = \mathcal{E}(m_i, r_i)$ for all selected i 's.

Depending on the size of x , this method is a simple way to confirm that the $n - x$ remaining ciphertexts are highly likely to be correctly encrypted as well, because of the random nature V selects the ciphertexts for decryption. We will use this protocol when verifying the correctness of the ballots, as it is a simple and effective way to do so.

6.4 Mixnet Shuffle

We will now summarise the basics of a decrypting mixnet based on a public key cryptosystem $(\mathcal{K}, \mathcal{E}, \mathcal{D})$.

Players: Electoral commission EC, Mixnet servers \mathcal{M}_i for $i = 1, \dots, k$.

Key generation:

1. EC chooses the public and secret keys $(pk_i, sk_i) \leftarrow \mathcal{K}$ for $i = 1, \dots, k + 1$.
2. EC sets $pk = (pk_1, \dots, pk_{k+1})$.

Encryption: For randomness $r = (r_1, r_2, \dots, r_{k+1})$, we encrypt m with:

$$\mathcal{E}_M(pk, m; r) = \mathcal{E}(pk_1, \mathcal{E}(pk_2, \dots, \mathcal{E}(pk_{k+1}, m; r_{k+1}); \dots; r_2)r_1)$$

Mixnet mixing For input $c_1^{(i-1)}, \dots, c_N^{(i-1)}$, each mixnet server mixes as follows, for $l = 1, \dots, N$:

1. $\tilde{c}_l^i \leftarrow \mathcal{D}(sk_l, c_l^{i-1})$
2. Choose a random permutation π .
3. $c_l^i \leftarrow \tilde{c}_{\pi(l)}^{(i)}$
4. Prove that $c_1^{(i-1)}, \dots, c_N^{(i-1)}$ decrypt to a permutation of $c_1^{(i)}, \dots, c_N^{(i)}$

Decryption Finally, the EC decrypts with

$$v_l \leftarrow \mathcal{D}(sk_{k+1}, c_l^{(k)}),$$

and outputs l proofs of correct decryption.

The zero-knowledge proofs of correct shuffles are not included here but for further reading there is, for example, Bayer and Groth's 'Efficient Zero-Knowledge Argument for Correctness of a Shuffle'. [11]

7 Fiat-Shamir Heuristic

The Fiat-Shamir heuristic, coined in 1986 by Fiat and Shamir, is a technique for replacing the interactive part of a proof of knowledge, with a non-interactive random oracle. It is only applicable to public-coin interactive proofs, i.e. proofs in which the Verifier's random choice is made public, as it is in our protocols. We will describe how the random oracle model works, in theory and in practice, and then give an example of how this applies to a proof of knowledge.

7.1 Random Oracle Model

A random oracle is a theoretical oracle, used to prove a level of security in cryptosystems. From any unique input they respond with a truly random response. This response is taken uniformly from the output domain such that if the same input is submitted again, the same response will be given from the random oracle. It can be seen as a map from all the possible inputs to their fixed random

responses. This is a theoretical model as no function given by a finite algorithm may implement such an oracle, according to the Church-Turing thesis.

However, for our purposes we can replace the random oracle model with a cryptographic hash function, as this gives a sufficient level of security for our security goals. Hash functions should act as much as possible like a random oracle, whilst still being computable and deterministic. We will not define what specific hash function for our proofs, but instead refer to a generic one, \mathcal{H} .

7.2 Application to P_n

We will now apply the Fiat-Shamir Heuristic technique to the Protocol for n -th powers we described in the previous section, P_n . Note that the non-interactive version of P_n we demonstrate is the same except for step 2, where instead of the verifier interacting and choosing β , the prover chooses it with a hash function.

Non-Interactive Version of the Protocol for n -th powers

1. P chooses ρ , sends $\alpha \leftarrow \rho^n$ to V.
2. P chooses $\beta \leftarrow \mathcal{H}(\alpha, \dots)$ from a hash function.
3. P calculates $\gamma \leftarrow \rho r^\beta \bmod n$ and sends to V.
V verifies that $\gamma^n = \alpha c_0^\beta \bmod n^2$. If this is the case, and c_0, α, γ are prime to n , it shows that c_0 is an n 'th power.

So with the use of the Fiat-Shamir Heuristic we have converted an interactive proof of knowledge to a non-interactive version. We can use this to convert a SHVZK proof into a NIZK (Non-Interactive Zero Knowledge) proof. We apply this method, as demonstrated, to our SHVZK proof of correct decryption to obtain a NIZK proof of correct decryption, as the change in this step makes no difference to the proof. As such, we now have a NIZK proof of correct decryption, which applies to Paillier encryption. Since the protocol is now non-interactive, we will refer to it as the (ZK) proof of decryption later on.

Candidate List	Your Vote	Candidate List	Your Vote
Karl		Thomas	
Rosa		Helen	X
Thomas		Rosa	
Helen	X	Karl	
	F8A2kL		v5EA7M
<i>Destroy this Side</i>	<i>Keep this Side</i>	<i>Destroy this Side</i>	<i>Keep this Side</i>

Table 3: Two examples of *Prêt à Voter* ballots, with randomised candidate orders, both with votes for the same candidate.

Part III

Choices for subprotocols of two *Prêt à Voter* Variations

In this section we will specify the two variations we are going to analyse and give security proofs for in the final section. For each subprotocol in the variation, we may mention a few options that could be implemented, then present which one we will choose for our variants. We will refer to these two variants as *Paillier Prêt à Voter* and *Prêt à Voter with Mixnets*.

8 Implementation of *Paillier Prêt à Voter* and *Prêt à Voter with Mixnets*

8.1 Ballot Generation

Assume that there are R candidates in the election. The ballots in *Prêt à Voter Classic* are created in the form shown in Table 3. A randomised candidate list is generated on the left hand side, whilst the ballot boxes and an encryption key which encrypts the candidate order is on the right hand side. There are multiple ways to do this.

For our variants, we create a player known as the Electoral Commission, EC, that generates the keys to be used in encrypting and decrypting the ballots. They send the keys to a player Ballot Generator, BG, which creates the ballots.

Since we want to prove two simple variations, we simplify the ballot encryption slightly, so that the encryption doesn't need to be combined with the mark in a separate protocol. Instead, we generate ballots with a different ciphertext, c_i for each candidate, A_i for R candidates, $i = 1, \dots, R$, as demonstrated in Table 4. Then the ballots are sealed in indistinguishable envelopes so that the candidate list order for each ballot is kept a secret only known to the voter who selects that ballot.

Paillier Prêt à Voter For this variant, each ballot is encrypted using the Paillier cryptosystem, such that $c_i = g^{A_i} \cdot r_i^n \bmod n^2$ for a public key (n, g) and random elements r_i for $i = 1, 2, \dots, R$, as described in the Tools section. We will assign each candidate A_i to be L^i in order to accomodate simple homomorphic tabulation in the counting stage.

This is a useful cryptosystem for voting system purposes for a couple of reasons. It is an additive homomorphic cryptosystem, so the counting stage is simple to perform, as homomorphic tabulation can be used. Also, because of the random element in the encryption, a candidate’s encryption looks different for each ballot, so if an adversary can see multiple ballots they cannot easily see which ciphertext belongs to which candidate.

Prêt à Voter with Mixnets Mixnets can be done with any public key cryptosystem, $(\mathcal{K}, \mathcal{E}, \mathcal{D})$. To keep our security proof more general, we will not specify this cryptosystem, and instead just assume it is secure and the secret keys do not leak. The key generation and encryption is done as described in the Tools section. We can assign each candidate A_i to be represented simply by i , giving $c_i = \mathcal{E}_M(pk, A_i; r_i)$ for $i = 1, 2, \dots, R$.

8.2 Verifying the correctness of the ballots

This is another area in which there are several methods to choose from. For example in the *vVote* variation we briefly described earlier, the voter Alice may verify the ballots ‘live’, and similarly in the *Scratch & Vote* variation the voter may scratch a spare ballot to verify they are generated correctly. These are based on individual voters verifying as they go however, so for a more systematic verification process we will implement the Cut-and-Choose protocol. We will use it in the same way for both the Paillier and Mixnet variations, and assign the task to a player Ballot Verifier, BV.

Cut-and-Choose We use Cut-and-Choose, as described in the Tools section, as it is a simple and effective method for verifying the ballots are generated correctly. The protocol goes as outlined in the Tools section, where each ciphertext is checked for each ballot.

Assume there are $< L$ voters. For this instance, we will check half the ballots, so therefore, there must be at least twice as many ballots generated as are needed, say $2L$ ballots in this case. Then, using a random process, this pile of ballots is divided into two piles say pile A and pile B , of L ballots each. Then the ballot verifiers can decrypt pile A , and confirm that the encrypted ciphertexts printed on the right hand side corresponds correctly to the candidates on the left hand side.

For the *Paillier Prêt à Voter* variant, verification involves receiving the randomness used from the Ballot Generation, r_i , and confirming that

$$c_i = g^{A_i} \cdot r_i^n \bmod n^2,$$

for all L ballots. Similarly, for the *Prêt à Voter with Mixnets* variant, the randomness r_i is received and the verifiers check that

$$c_i = \mathcal{E}_M(pk, A_i; r_i)$$

for all L ballots.

If all L of these ballots are accurate, the ballots have been verified with a high certainty. Of course, there is a chance the only faulty ballots are in pile B , but this is extremely unlikely assuming there are multiple faulty ballots. For example, if there are n faulty ballots, say $F = f_i : i = 1, \dots, n$, then the chance of at least one of these being in pile A increases exponentially with the number of faulty ballots, n , at

$$Pr(F \cap A \neq \emptyset) = 1 - 1/2^n$$

since each faulty ballot f_i is randomly allocated, so has a chance of $\frac{1}{2}$ to be in pile A .

So assuming pile A contains no faulty ballots, we can say with high certainty that pile B also has no, or insignificant, faulty ballots. So finally we discard pile A , as its security has been compromised, since the verifiers have seen the candidate orders, and we use pile B as our verified ballots for use in the protocol.

We chose Cut-and-Choose to verify our ballots as it is very simple to implement in any circumstance, and gives a high probability that the ballots were generated correctly. In fact, Cut-and-Choose can be applied to any voting system with printed ballots. The main problem is that when a very large number of ballots are being produced, the number that must be checked to give us a low chance of faults gets very large as well, and this may be time-consuming.

8.3 Vote Casting

The vote casting process is more of a practical choice than a cryptographic choice, so we will be brief in our discussion. The most important aspect is voter booth privacy, as the security of the protocol relies heavily on the order of each voter's candidate list being known only to that voter. As such, we require a private voter booth and a system to destroy the left hand side of the ballot in it. This is the cryptographic requirement, the practicalities of such a system can be decided by the officials running the election, as long as the intact candidate orders cannot be recovered and linked to particular voters. Once the left hand side is destroyed, the right hand side is simply submitted to a ballot box.

8.4 Vote Processing and Counting

The vote processing and counting include any subprotocols that occur once the votes have been marked, including any shuffling, decryption and tallying of the votes. For the *Paillier Prêt à Voter* variant this just includes counting and decryption, whilst the *Paillier Prêt à Voter* variant shuffles the votes, then decrypts them and the counting is trivial.

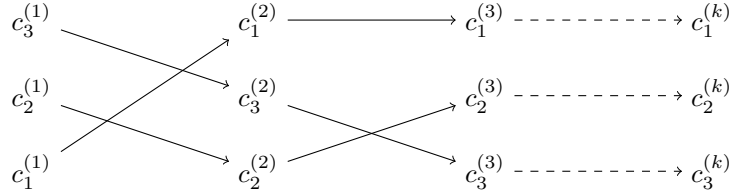


Figure 2: Mixnet Shuffle

Paillier Prêt à Voter The main advantage of the Paillier encryption method is that it is an additive homomorphic cryptosystem, so homomorphic tabulation can be used in counting the votes. Since we have encrypted votes c_i for $i = 1, \dots, L$, the Electoral Commission computes $c = \Pi c_i$, and using the secret keys, decrypts this to $m = \sum t_j L^j$ where t_j represents the coefficient of L^j , i.e. the number of votes that candidate A_j received. So m is posted to the Bulletin Board, BB, as the final result of the election.

Prêt à Voter with Mixnets The Mixnet shuffle is a method that was first described in 1981 by David Chaum, and is used to take in messages, shuffle them, and send them back out in a random order. The Mixnet shuffle process is very useful for voting protocols, as the shuffling ensures that the voters keep their privacy. Another relevant advantage of the Mixnet shuffle is that it can take messages from several ‘players’. For example, several voting booths could send their collected votes to a central Mixnet server.

There are many variations, for example reencrypting Mixnets which would be suitable for Paillier, but these are complex. Instead for this variation we will use a decrypting Mixnet for k Mixnet servers M_1, \dots, M_k . We described this protocol in the Tools section so we will not repeat the details. In summary, the Electoral Commission first sends the relevant secret keys to the Mixnet servers. They take as input the encrypted ciphertexts, then they each shuffle the order of the ciphertexts using random permutations, in order from 1 to k . At each step they post the output along with a proof of correct mixing to the BB. Finally, M_k sends $c_1^{(k)}, c_2^{(k)}, \dots, c_N^{(k)}$ to the EC that decrypts it to v_1, \dots, v_N and posts this, along with the proof of decryption to the BB. The shuffling method, for 3 ciphertexts, is demonstrated in Figure 2.

8.5 Auditing

Auditing is an important process that aims to ensure that individual voters can check that their votes were cast as intended. Also, public auditors should be able to check the various subprotocols, and verify the proofs for correct encryption, mixing and decryption.

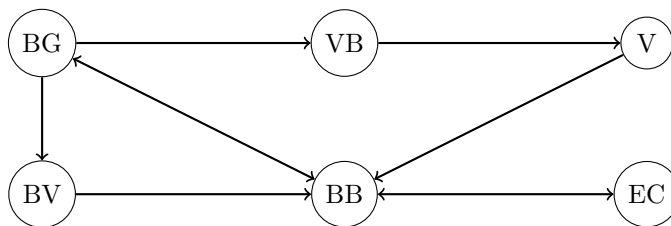


Figure 3: Player Interactions in Paillier Prêt à Voter

Paillier Prêt à Voter Since the submitted ballots are made public on the BB, each player may check that their ballot was recorded correctly. As these ballots are just the right hand side, and only the voters know their corresponding candidate order, this is enough for voters to audit their personal vote. In addition, their privacy isn't being threatened since the ballots posted are encrypted and aren't linked to the voter.

For public auditors, who aim to check the accuracy of the entire protocol, a zero-knowledge proof of correct decryption is needed for the Paillier system. We described this proof for the n -th power protocol in the Tools section, which shows that a ciphertext is an n -th power, and hence correctly decrypts. We also showed it was non-interactive zero-knowledge, by using the Fiat-Shamir Heuristic. Therefore, public auditors should be able to verify the election was carried out fairly.

Prêt à Voter with Mixnets Similar to the other variant, the submitted ballots are made public on the BB, so each player may confirm their own ballot was recorded correctly.

In order for public auditors to check each step, a zero-knowledge proof of correct mixing is provided at each step of the Mixnet shuffle process, to show that each set of ciphertexts is indeed a permutation of the last, without revealing the permutation itself. Also, a zero-knowledge proof of decryption is provided by the EC. These zero-knowledge proofs are out of the scope of this paper.

9 Player Interactions in *Paillier Prêt à Voter* and *Prêt à Voter with Mixnets*

We can split these systems into several subprotocols, as we will list in our description of the processes. We briefly explain which parts of the diagram correspond to each part of the protocols.

Paillier Prêt à Voter In Figure 3 we demonstrate the interconnected players in the *Paillier Prêt à Voter* protocol.

Ballot Generation uses keys generated by the Electoral Commission, EC, then is carried out in BG. Ballot Correctness verification is an interaction between

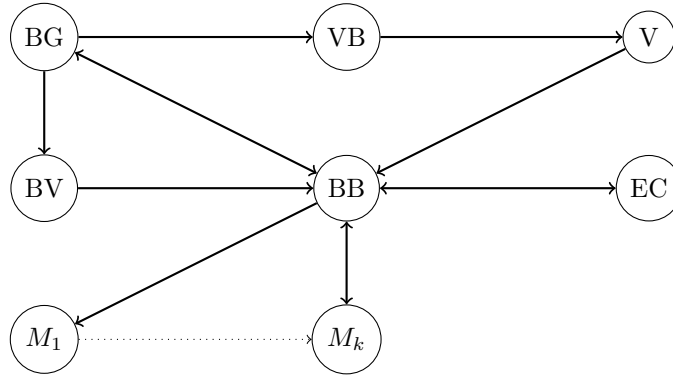


Figure 4: Player Interactions in Prêt à Voter with Mixnets

BG and some Ballot verifiers, BV, followed by BG sending the verified ballots to the voting booth VB. Marking is an interaction between VB and V, the voter. Submission is a one way interaction between V and BB, the Bulletin Board. Decryption takes place in the EC, which sends the results and proofs to the BB. Tallying and posting of results is an interaction within the BB. Finally, auditing uses the results posted publicly on the BB.

Prêt à Voter with Mixnets In Figure 4 we demonstrate the interconnected players in the *Prêt à Voter with Mixnets* protocol.

As you can see, the interactions are the same, with the addition of the Mixnet Shuffle that takes place between the BB and the Mixnet servers M_1 to M_k , which post each step and proofs of correct mixing to the BB as well, for the auditing stage.

Now that we have our different sections outlined, we briefly describe the players and processes for reference in the security proof.

9.1 Players

The players in the *Prêt à Voter* protocols are as follows:

1. **Electoral Commission, EC:** The Electoral Commission generates the public and secret keys for the Ballot Generator, and also performs the Decryption stage.
2. **Ballot Generator, BG:** The Ballot Generator is allocated Ballot Generation, and is responsible for generating and printing $2L$ ballots.
3. **Ballot Verifiers, BV:** The Ballot Verifiers are in charge of Ballot Correctness verification.
4. **Voters, V:** The voters are allocated the tasks of Marking and Submission, as well as the choice to be involved in auditing by checking their vote was

Candidates	Vote	Encryption
A_1		c_1
A_2		c_2
...		...
A_j		c_j

Table 4: Encrypted Ballot

cast as expected. Only they can confirm this as only they have seen the left hand side of their ballot, so can be sure their vote was cast correctly, just from seeing the submitted right hand side.

5. **Bulletin Board:** The Bulletin Board is allocated the Tallying and Posting of results, including the proofs for decryption and in the case of the Mixnet variation, proofs of correct mixing.
6. **Auditors:** These independent observers perform Auditing, to ensure the voting procedure was performed fairly and correctly, and votes are decrypted as expected. In the case of the Mixnet variant, they also check that the Mixnets shuffle the ciphertexts correctly.
7. **Mixnet Servers, M_1, \dots, M_k :** The Mixnet servers are players only in the variant with Mixnets, and are responsible for the Mixnet Shuffles, as well as for posting each step of this to the Web Bulletin Board so the accuracy of the shuffle can be checked in Auditing.

9.2 Description of the Processes

We will describe each stage for the two variants now, demonstrating the sub-protocols we have described and summarising this part before we move onto the security proofs.

Paillier Prêt à Voter

1. **Ballot Generation** The EC generates the public and secret keys, and sends them to the BG. For $< L$ voters, $2L$ ballots are generated as shown in Table 4, where c_i are encrypted with the Paillier cryptosystem as described earlier. At random, L generated ballots are sent to the BV to be verified, and the remaining L are sent to the VB.
2. **Ballot Verification** The relevant secret keys are sent from the EC to the BV, and the ballots are verified, by checking that the candidates encrypt to the corresponding ciphertexts on the ballots, as expected. Assuming the ballots are correct, we discard this half of the ballots as their privacy is corrupted, and confirm that the other L ballots are verifiably correct.

3. **Marking and Submission** The voters V enter VB , each take a ballot and mark their chosen candidate on the appropriate row. Then, in the secrecy of the booth they tear the two halves apart and discard the LHS with the candidate list, and leave the booth with the RHS. They submit this vote, which is equivalent to c_i , to the Bulletin Board. The BB then submits the entire collection of submitted votes to the EC.
4. **Decryption** The EC, using the secret keys, decrypts and tallies the encrypted votes with homomorphic tabulation, to obtain the complete results of the election, and posts them to the BB, along with a proof of correct decryption, which we described in the Tools section.
5. **Tallying and posting of results** The BB is made public and anyone may audit the entire process, by checking the proof of decryption. Individual voters may verify that their vote was counted by finding their submitted ballot on the BB.

Prêt à Voter with Mixnets

1. **Ballot Generation** The EC generates the public and secret keys, and sends them to the BG. For $< L$ voters, $2L$ ballots are generated as shown in Table 4, where c_i are encrypted by a public key cryptosystem, $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ as described earlier. At random, L generated ballots are sent to the BV to be verified, and the remaining L are sent to the VB.
2. **Ballot Verification** The relevant secret keys are sent from the EC to the BV, and the ballots are verified, by checking that the candidates encrypt to the corresponding ciphertexts on the ballots, as expected. Assuming the ballots are correct, we discard this half of the ballots as their privacy is corrupted, and confirm that the other L ballots are verifiably correct.
3. **Marking and Submission** The voters V enter VB , each take a ballot and mark their chosen candidate on the appropriate row. Then, in the secrecy of the booth they tear the two halves apart and discard the LHS with the candidate list, and leave the booth with the RHS. They submit this vote, which is equivalent to c_i , to the Bulletin Board. The BB then submits the entire collection of submitted votes to the first Mixnet server.
4. **Mixnet Shuffle** The Mixnet Shuffle, as described earlier, takes the votes, shuffles them with a random permutation, then sends it to the next server for all the Mixnet servers $M_k, j = 1, \dots, k$. For each server there is a proof of knowledge such that it can be verified later that the shuffle procedure was done correctly. Finally, the fully shuffled votes, along with the proofs, are sent to the BB.
5. **Decryption** The BB sends the encrypted messages to the EC. The EC decrypts the encrypted votes using the secret keys for the public key cryptosystem, to obtain the complete results of the election, and posts them to the BB, along with a proof of correct decryption.

6. **Tallying and posting of results** The BB is made public and anyone may audit the entire process, by checking the proof of decryption and the proofs of mixing. Individual voters may verify that their vote was counted by finding their submitted ballot on the BB.

The exact calculations carried out by each player will be listed for each variant in the Security Proof section.

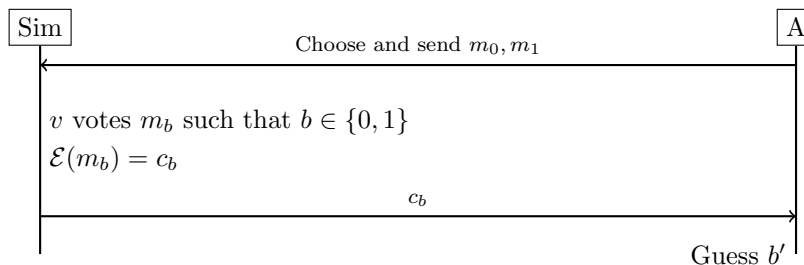


Figure 5: Privacy

Part IV

Security Proofs for *Paillier Prêt à Voter* and *Prêt à Voter with Mixnets*

In this section we will prove that the two variations, Mixnet and Paillier Prêt à Voter, are secure. First we set up what it means to be secure for our purposes by defining privacy, coercion-resistance and verifiability. Then, we provide security proofs for these. Since these proofs have similarities, there will be progressively less detail in these proofs. Finally we will provide some remaining attacks that have not been addressed, and could be considered in future papers.

10 Security Goals

First we must define what we mean by security, and in what context *Prêt à Voter* is secure. We will define privacy, coercion-resistance and verifiability, and say that a protocol is secure if it has all these properties.

Definition 10.1. A system has **privacy** if the information transmitted from an individual sender is not accessible by attackers, i.e. if Alice votes for m , submitting c as her ballot, an attacker cannot determine who she voted for without access to the secret key.

Privacy can be demonstrated in Figure 5, in which the system has privacy if the adversary's advantage in guessing b is negligible, i.e.,

$$2Adv_{\mathcal{A}} = |Pr[b = b'] - \frac{1}{2}|$$

is negligible.

In our protocol, privacy is potentially under threat in the stages Ballot Generation, Voting and Ballot Submission. So to be convinced our protocol has

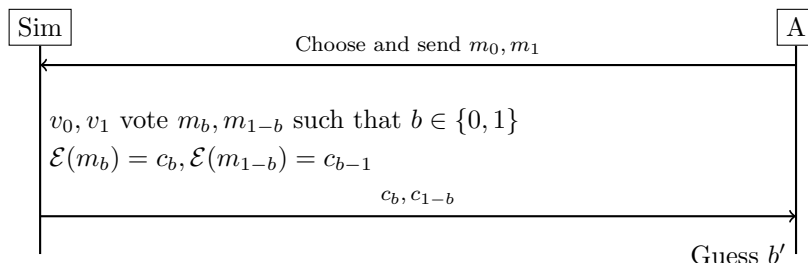


Figure 6: Coercion-Resistance

privacy, there cannot be leaks in any of these stages which would give the adversary an advantage, for example if they obtain a secret key.

Definition 10.2. A system has **Coercion-Resistance** if, allowing the attacker to demand coerced voters vote a certain way, the attacker still cannot determine whether the voter complied. [7]

Coercion-Resistance is demonstrated in Figure 6. Since in the extreme case that the adversary tells the voter to vote for someone who receives no votes, coercion-resistance clearly fails, we must assume that each candidate gets at least one vote, and form a diagram as above.

The adversary coerces a voter to vote m_0 , and the voter v_0 either votes for m_0 or m_1 , with voter v_1 voting the other way. If the system is such that the adversary has negligible advantage in guessing which voter voted for m_0 , then we say the system is coercion-resistant, i.e. if

$$2Adv_{\mathcal{A}} = |Pr[b = b'] - \frac{1}{2}|$$

is negligible then the system is coercion-resistant.

Definition 10.3. A system is **verifiable** if a player sending a message is able to prove that their message was received, encrypted and decrypted correctly.

This definition is self-explanatory and we will apply it specifically to our Paillier protocol in the verification proof.

So we will have defined the protocol as meeting our security goals if it proves to have the properties of privacy, coercion-resistance and verifiability. We will prove that *Paillier Prêt à Voter* and *Prêt à Voter with Mixnets* have privacy, then show that *Paillier Prêt à Voter* has coercion-resistance and verifiability. The proofs of coercion-resistance and verifiability for *Prêt à Voter with Mixnets* are not included, as they are quite similar to the equivalent proofs for *Paillier Prêt à Voter*.

LHS	RHS
$\pi_i(1)$	$c_{i,\pi_i(1)}$
$\pi_i(2)$	$c_{i,\pi_i(2)}$
\dots	\dots
$\pi_i(R)$	$c_{i,\pi_i(R)}$

Table 5: Generated *Prêt à Voter* ballot with Paillier encryption.

11 Security Proofs for Privacy

In this section we will prove using the same method that both variations of *Prêt à Voter* have privacy, with the assumption that the players are honest. We do this by setting up several games, proving indistinguishability between them and setting up Real or Random simulations to show that the adversary has negligible advantage in their attack.

11.1 Paillier *Prêt à Voter*

In Figure 7 we demonstrate how this variant is run with the players and what information is passed between them. We will now explain the specific steps in the protocol.

Key Generation The Electoral Commission, EC, runs Paillier key generation, posts the public key (n, g) , and keeps the secret key (λ, μ) .

Ballot Generation BG Creates ballot $\#i$ for R candidates as follows:

$$c_{i,j} = r_{i,j}^n g^{L^j} \text{ for } j = 1, 2, \dots, R,$$

where $r_{i,j}$ random elements $\in \mathbb{Z}_{n^2}^*$.

Also, choose a random permutation π_i on $\{1, 2, \dots, R\}$.

The ballot is generated as shown in Table 5.

When ballot $\#i$ is chosen for audit by BV: BG gives $r_{i,1}, r_{i,2}, \dots, r_{i,R}$ to BV.

BV computes $\tilde{c}_{i,j} = r_{i,j}^n g^{L^j}$ for $j = 1, 2, \dots, R$ and checks that $c_{i,j} = \tilde{c}_{i,j}$ for $j = 1, \dots, R$.

Voting Booth The voter receives ballot $\#i$, and marks $\#i$ in position v_i . The marked RHS is recorded on the bulletin board and let $c_i = c_{i,v_i}$.

Counting The EC computes $c = \Pi c_i$ and decrypts to get $m = \sum t_j L^j$. They prove in Zero Knowledge that m is the correct decryption of c , i.e. that cg^{-m} is an n -th power. We demonstrated this proof in the Tools section.

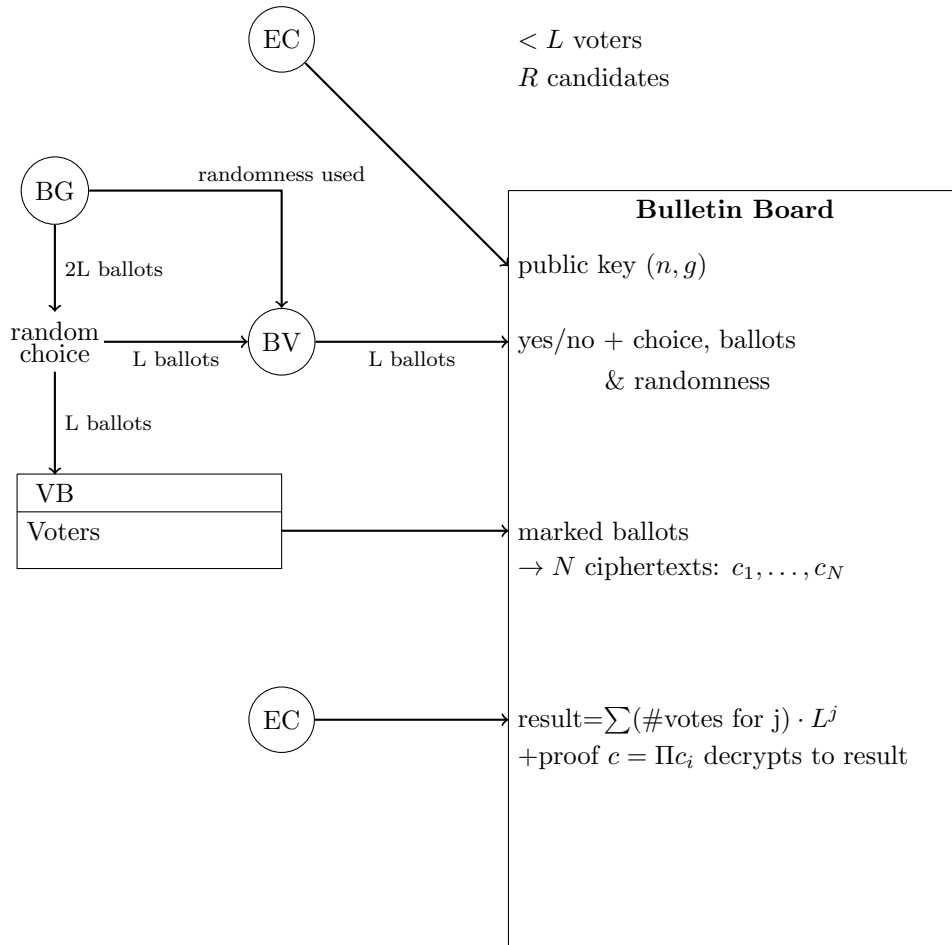


Figure 7: Systems Perspective of the Protocol using Paillier encryption

11.1.1 Privacy Game

We will now define a Privacy game, for use as the basis of our security proof. In it, the Simulator plays all participants, whilst the Adversary sees everything that is written to the bulletin board.

1. Simulator runs key generation. Flips a coin b .
2. Simulator runs ballot generation, then ballot verification.
3. Adversary specifies votes $(v_{i,0}, v_{i,1})$ for $i = 1, 2, \dots, N$, with $\sum L^{v_{i,0}} = \sum L^{v_{i,1}}$.
4. Simulator has every voter mark their ballot with v_i, b and submits it.
5. Simulator runs counting.
6. Adversary guesses b' .

Now let E represent the event that $b = b'$. We aim to prove, for privacy, that

$$2Adv_{\mathcal{A}} = |Pr[E] - \frac{1}{2}|$$

11.1.2 Security Proof

Let E_k be the event in Game k that $b = b'$.

Game 0 The Privacy game as we just described.

$$Pr(E_0) = Pr(E)$$

Game 1 In this game we use the Zero Knowledge simulator to create the decryption proof. The change cannot be observed by the adversary because conversations output by the simulator are indistinguishable from real conversations.

$$Pr[E_1] = Pr[E_0]$$

Game 2 Stop decrypting c and instead use $\sum L^{v_i, b}$ as the result. This works because Paillier is additively homomorphic, so since the decryption of c_{i, v_i} is $L^{v_i, b}$, the decryption of c is $\sum L^{v_i, b}$. Again this is indistinguishable to the adversary.

$$Pr[E_2] = Pr[E_1]$$

Game 3 Choose the ballots that will be verified before generating the ballots. This changes nothing observable.

$$Pr[E_3] = Pr[E_2]$$

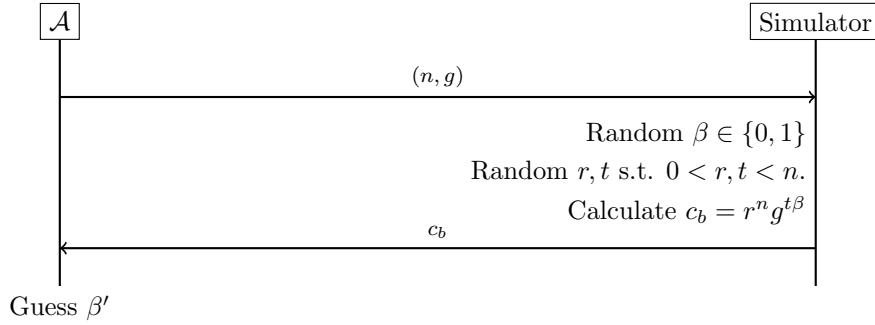


Figure 8: Paillier RoR Simulator

Game 4 Create the ciphertexts for the non-verified ballots as $c_{i,j} = r_{i,j}^n g^{s_{i,j}}$, where $s_{i,j}$ is random. If this changes anything, this is represented in a game of the adversary against real-or-random for Paillier encryption.

In this game, the adversary no longer gets information about which votes were actually encrypted, so no information about b . In order to determine $Pr[E_4]$ and the overall advantage, we create a game with a Real-or-Random adversary \mathcal{B} , based on the interaction of a Reduction Simulation and a Paillier RoR Simulator.

Paillier RoR Simulator This Paillier RoR Simulator, shown in Figure 8 takes as input the public key (n, g) . Then it flips a coin to decide β . It either outputs the correct value for c_b to be used in the encryption (if $\beta = 0$) or one that will eventually encrypt a random message instead ($\beta = 1$). The protocol goes as follows:

1. The public key (n, g) is generated by the adversary \mathcal{A} and sent to the Simulation.
2. Generate β, t, r at random, such that $\beta \leftarrow \{0, 1\}$ and $0 < r, t < n$.
3. Calculate c_b in the form $c_b = r^n g^{t\beta} \bmod n$.
4. Outputs c_b to the Adversary, \mathcal{A} .

The Decisional composite residuosity assumption, DCRA, is that given n, z , it is hard to determine if z is an n -residue modulo n^2 . In other words it is hard to know whether y exists such that

$$z \equiv y^n \bmod n^2$$

By the Decisional Composite Residuosity assumption, it is hard to distinguish whether c_0 is an n -th power, so the advantage of the adversary in guessing $\beta' = \beta$ correctly is negligible.

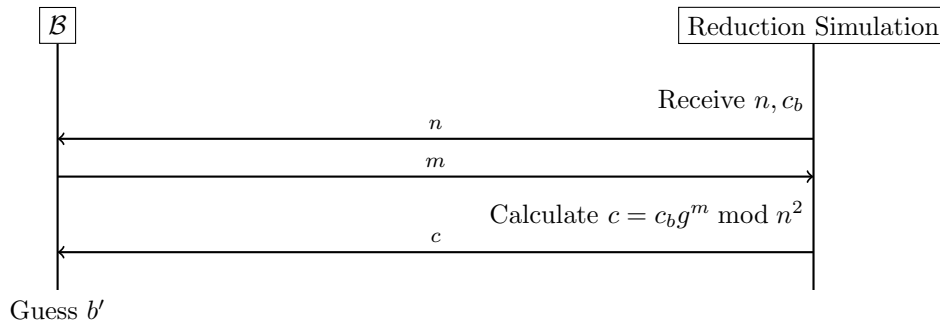


Figure 9: Reduction Simulation

Reduction Simulation This Simulation is known as the Reduction, shown in Figure 9. It takes (n, g) , c_b and calculates the complete ciphertext from them, then sends it to the adversary who guesses if it was the correct encryption or not. The protocol goes as follows:

1. Public key (n, g) , and c_b are taken as input from the DCRA Simulation.
2. The public key, n , is sent to the adversary, \mathcal{B} .
3. The adversary returns a message m
4. The reduction calculates $c = c_b g^m \bmod n^2$ and sends it to the adversary.
5. The adversary guesses b' and outputs it.

So on its own this simulation does not tell us much, but if we use it with the RoR simulation as input, we create a game in which our adversary receives either a correctly encrypted ciphertext of a message m , or an encryption of a random message. In this case the random message would be $s = t + m$ such that t is chosen randomly.

RoR Oracle The RoR Oracle is illustrated in Figure 10, where the dotted outline shows where the RoR Simulator is used, and the filled outline shows where the Reduction Simulation is used, both as described above with minor differences. Specifically, the adversary in our specified RoR Simulator is played by the Reduction Simulation, and instead of the Reduction Simulation starting with $(n, g), c_0$, it instead starts by generating (n, g) , then sends it to the RoR Simulator and receives c_0 in return.

Now we have simplified this interaction to an RoR oracle, which either correctly encrypts the votes or encrypts random messages, depending on the flip of a coin, we can demonstrate how it is used in the context of the wider protocol. In Figure 11 the adversary \mathcal{B} sees everything that is published to the bulletin

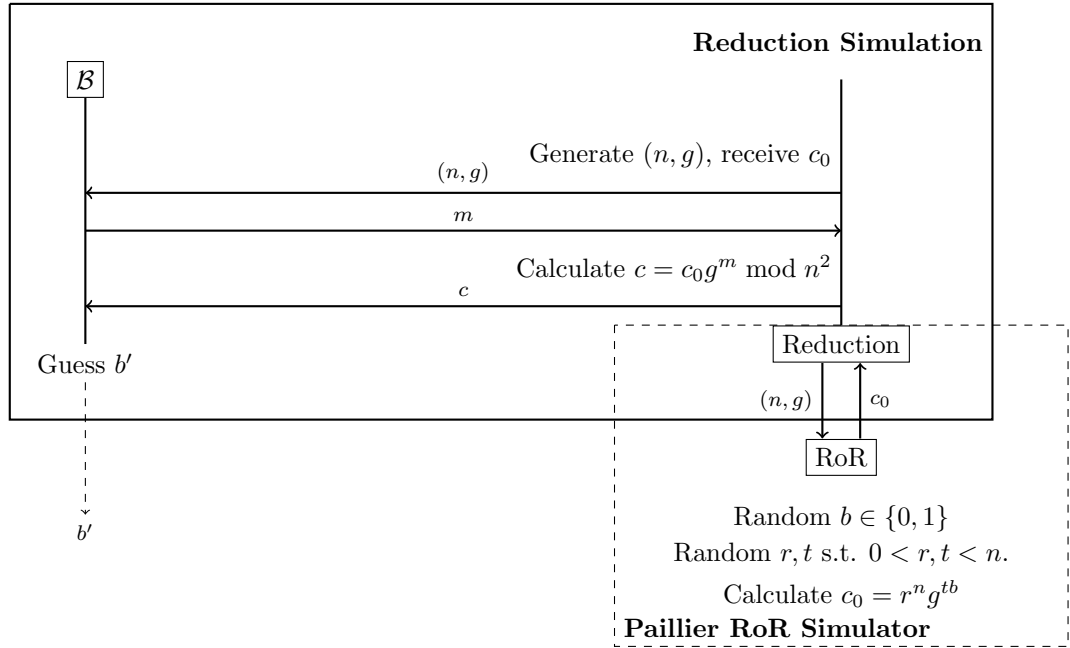


Figure 10: How the simulations interact

board, BB. The Paillier RoR Simulator is as described before, whilst the adversary \mathcal{A} represents the adversary in the Reduction simulation, submitting votes $(v_{i,0}, v_{i,1})$. Using the information from the bulletin board, they make a guess $b' \leftarrow 0, 1$ as to whether the votes $(v_{i,0})$ or $(v_{i,1})$ were encrypted. Then, the RoR adversary \mathcal{B} guesses $\beta' = 1$ if $b = b'$ and $\beta' = 0$ otherwise. This essentially is to determine if the use of the RoR oracle gives any extra information to the adversary.

We now have the required set-up to prove the security of this variant of the protocol.

Theorem 11.1. *Paillier Prêt à Voter has voter privacy*

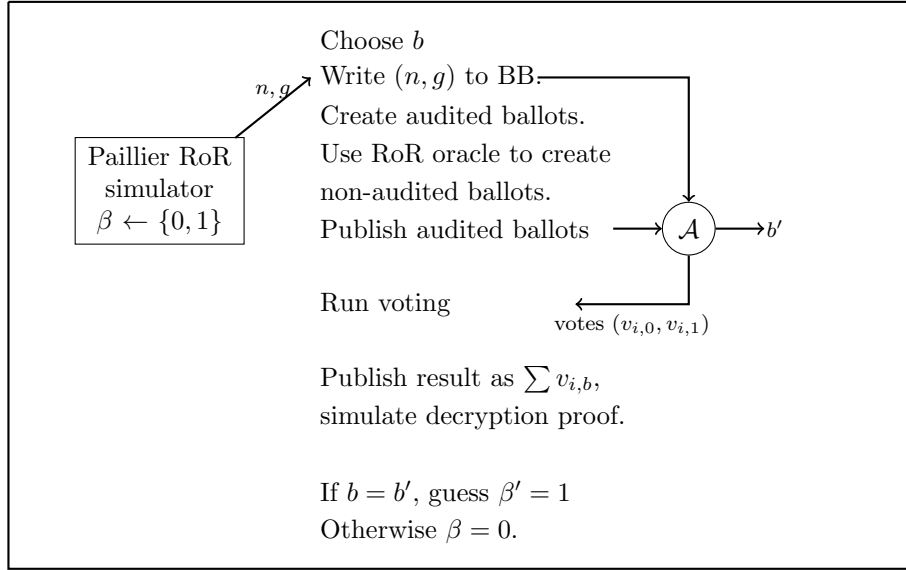


Figure 11: Paillier RoR, Adversary \mathcal{B}

Proof. So with respect to this game, we can calculate the advantage of \mathcal{B} as:

$$\begin{aligned}
 Adv_{\mathcal{B}}^{Paillier} &= |Pr[\beta' = \beta] - \frac{1}{2}| \\
 &= |Pr[\beta' = 1] \cap Pr[\beta = 1] + Pr[\beta' = 0] \cap Pr[\beta = 0] - \frac{1}{2}| \\
 &= |Pr[\beta' = 1 \mid \beta = 1] \cdot Pr[\beta = 1] + Pr[\beta' = 0 \mid \beta = 0] \cdot Pr[\beta = 0] - \frac{1}{2}| \\
 &= \frac{1}{2} |Pr[b = b' \mid \beta = 1] + Pr[b \neq b \mid \beta = 0] - 1| \\
 &= \frac{1}{2} |Pr[E_4] + (1 - Pr[E_3]) - 1| \\
 &= \frac{1}{2} |Pr[E_4] - Pr[E_3]|
 \end{aligned} \tag{2}$$

Since when $\beta = 0$, the RoR simulation exactly runs Game 3, and when $\beta = 1$ it runs Game 4, this is how we get the identities

$$Pr[b = b' \mid \beta = 1] = Pr[E_4]$$

$$Pr[b \neq b \mid \beta = 0] = 1 - Pr[E_3]$$

So this gives us overall

$$|Pr[E_4] - Pr[E_3]| = 2Adv_{\mathcal{B}}^{Paillier}$$

LHS	RHS
$\pi_i(1)$	$c_{i,\pi_i(1)}$
$\pi_i(2)$	$c_{i,\pi_i(2)}$
\dots	\dots
$\pi_i(R)$	$c_{i,\pi_i(R)}$

Table 6: Generated *Prêt à Voter with Mixnets* ballot.

We have shown that the games are indistinguishable, giving us:

$$Pr[E_3] = Pr[E_2] = Pr[E_1] = Pr[E_0] = Pr[E]$$

Also, we learnt from the DCRA assumption when constructing the RoR simulator that there is negligible advantage in guessing whether $\beta = \beta'$. Since this is equivalent to running game 4, this gives us that $Pr[E_4] = \frac{1}{2}$. Therefore we have, as required for privacy,

$$Pr[E] - \frac{1}{2} \leq 2Adv_{\mathcal{B}}^{Paillier}$$

□

11.2 Prêt à Voter with Mixnet

In this section we will use a very similar proof to show that the variation of Prêt à Voter using Mixnet shuffles is also secure in terms of privacy. In Figure 12 we demonstrate how this variant is run with the players and what information is passed between them.

Key Generation The Electoral Commission, EC, runs key generation and posts the public key pk , and keeps the secret keys sk .

Ballot Generation BG Creates ballot $\#i$ for R candidates as follows:

$$c_{i,j} = \mathcal{E}_M(pk, j; r_{i,j}) \text{ for } j = 1, 2, \dots, R,$$

where $r_{i,j}$ is randomness. Also choose a random permutation π_i on $\{1, 2, \dots, R\}$. The ballot is generated as shown in Table 6.

When ballot $\#i$ is chosen for audit by BV, BG gives $r_{i,1}, r_{i,2}, \dots, r_{i,R}$ to BV. BV computes $\tilde{c}_{i,j} = \mathcal{E}_M(pk, j; r_{i,j})$ for $j = 1, 2, \dots, R$ and checks that $c_{i,j} = \tilde{c}_{i,j}$ for $j = 1, \dots, R$.

Voting Booth The voter receives ballot $\#i$, and marks $\#i$ in position v_i . The marked RHS is recorded on the bulletin board and let $c_i = c_{i,v_i}$.

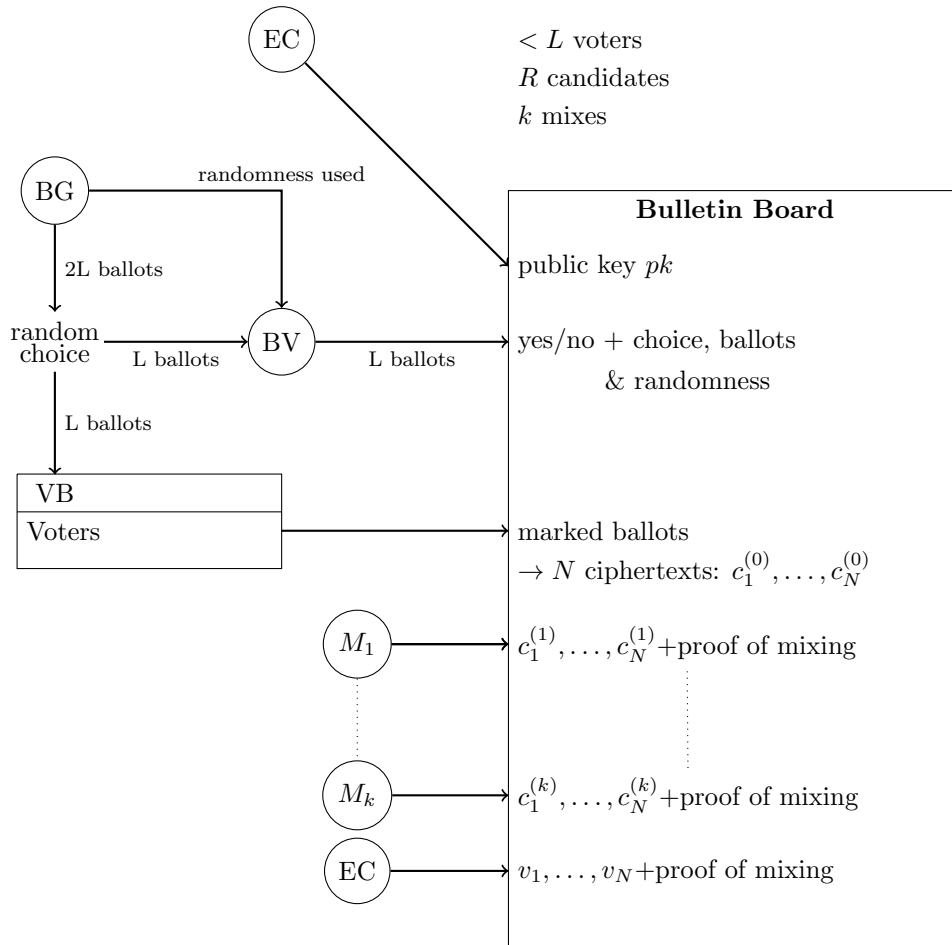


Figure 12: Systems Perspective of the Protocol using Mixnet

Counting The EC sends relevant secret keys to the mix servers M_1, \dots, M_k . M_i runs its Mixnet shuffle on $c_1^{(i-1)}, \dots, c_N^{(i-1)}$ and produces $c_1^{(i)}, \dots, c_N^{(i)}$ + the proof of mixing, for $i = 1, 2, \dots, k$. The output is posted on the BB. Then, the EC decrypts $c_1^{(k)}, \dots, c_N^{(k)}$ to v_1, \dots, v_N and posts this + proof of decryption to the BB.

11.2.1 Privacy Game

We will now define a Privacy game, for use as the basis of our security proof. In it, the Simulator plays all participants, whilst the Adversary sees everything that is written to the bulletin board.

1. Simulator runs key generation. Flips a coin b .
2. Simulator runs ballot generation, then ballot verification.
3. Adversary specifies votes $(v_{i,0}, v_{i,1})$ for $i = 1, 2, \dots, N$, with $\sum L^{v_{i,0}} = \sum L^{v_{i,1}}$.
4. Simulator has every voter mark their ballot with v_i, b and submits it.
5. Simulator runs counting.
6. Adversary guesses b' .

Now let E represent the event that $b = b'$. We aim to prove, for privacy, that

$$2Adv\mathcal{A} = |Pr[E] - \frac{1}{2}|$$

11.2.2 Security Proof

Let E_k be the event in Game k that $b = b'$.

Game 0 The Privacy game as we just described.

$$Pr(E_0) = Pr(E)$$

Game 1 In this game we use the Zero Knowledge simulator to create the decryption proof. The change cannot be observed by the adversary because conversations output by the simulator are indistinguishable from real conversations.

$$Pr[E_1] = Pr[E_0]$$

Game 2 Stop decrypting c and instead use v_{π_i} as the result, since we know π . Since $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key cryptosystem, we know that the decryption of c_i is $v_{\pi(i), b}$, because the correct decryption of c_{i, v_i} is $v_{i, b}$. Therefore this is indistinguishable to the previous game.

$$Pr[E_2] = Pr[E_1]$$

Game 3 Choose the ballots that will be verified before generating the ballots. This changes nothing observable.

$$Pr[E_3] = Pr[E_2]$$

Game 4 Create the ciphertexts for the non-verified ballots as $c_{i,j} = \mathcal{E}_M(pk, s_{i,j})$, where $s_{i,j}$ is random. If this changes anything, this is represented in a game of the adversary against real-or-random for $(\mathcal{K}, \mathcal{E}, \mathcal{D})$.

In this game, the adversary no longer gets information about which votes were actually encrypted, so there is no information about b . In order to determine $Pr[E_4]$ and the overall advantage, we create a game with a Real-or-Random adversary \mathcal{B} , based on the interaction of a Reduction Simulation and a $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ RoR Simulator.

RoR Simulator and RoR Oracle for $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ The RoR simulator for a public key cryptosystem is set up in the same way as for Paillier encryption, with equivalent simulations. Specifically, the messages are encrypted with the public key cryptosystem rather than with Paillier encryption. In the RoR simulator, c_0 is calculated as $m^{1-b} \cdot t^b$ so that the correct message m is sent when $b = 0$ and a random message t is sent when $b = 1$. The reduction simulation encrypts with the randomness r to give $c = \mathcal{E}_M(pk, c_0; r)$. We demonstrate in Figure 13 how this RoR simulator interacts with the adversary. Again the RoR oracle is the interaction of the RoR and Reduction simulation as described in Figure 10. Note that we only use the RoR oracle on the inner layer of the Mixnet shuffle.

We now have the required set-up to prove the security of this variant of the protocol.

Theorem 11.2. *Prêt à Voter with Mixnet has voter privacy*

Proof. So with respect to this game, we can calculate the advantage of \mathcal{B} as:

$$\begin{aligned}
Adv_{\mathcal{B}}^{Mixnet} &= |Pr[\beta' = \beta] - \frac{1}{2}| \\
&= |Pr[\beta' = 1] \cap Pr[\beta = 1] + Pr[\beta' = 0] \cap Pr[\beta = 0] - \frac{1}{2}| \\
&= |Pr[\beta' = 1 \mid \beta = 1] \cdot Pr[\beta = 1] + Pr[\beta' = 0 \mid \beta = 0] \cdot Pr[\beta = 0] - \frac{1}{2}| \\
&= \frac{1}{2} |Pr[b = b' \mid \beta = 1] + Pr[b \neq b \mid \beta = 0] - 1| \\
&= \frac{1}{2} |Pr[E_4] + (1 - Pr[E_3]) - 1| \\
&= \frac{1}{2} |Pr[E_4] - Pr[E_3]|
\end{aligned} \tag{3}$$

Since when $\beta = 0$, the RoR simulation exactly runs Game 3, and when $\beta = 1$ it runs Game 4, this is how we get the identities

$$Pr[b = b' \mid \beta = 1] = Pr[E_4]$$

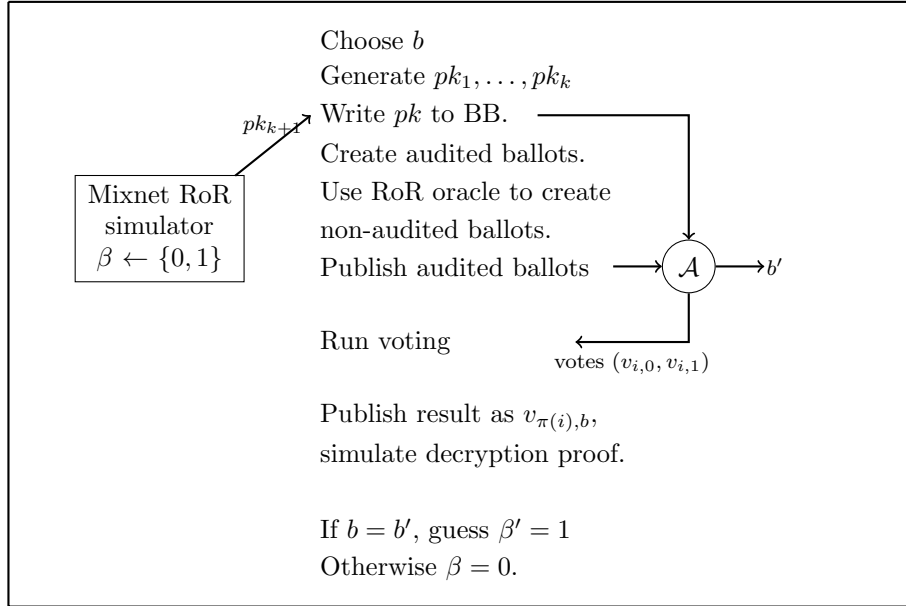


Figure 13: Mixnets RoR, Adversary \mathcal{B}

$$Pr[b \neq b | \beta = 0] = 1 - Pr[E_3]$$

So this gives us overall

$$|Pr[E_4] - Pr[E_3]| = 2Adv_{\mathcal{B}}^{Mixnet}$$

We have shown that the games are indistinguishable, giving us:

$$Pr[E_3] = Pr[E_2] = Pr[E_1] = Pr[E_0] = Pr[E]$$

Also, we learnt when constructing the RoR simulator that there is negligible advantage in guessing whether $\beta = \beta'$. Since this is equivalent to running game 4, this gives us that $Pr[E_4] = \frac{1}{2}$. Therefore we have, as required for privacy,

$$Pr[E] - \frac{1}{2} \leq 2Adv_{\mathcal{B}}^{Mixnet}$$

□

12 Proof of Coercion-Resistance

We now want to prove that our systems are secure against coercion. We simulate coercion as follows:

The adversary, \mathcal{A} chooses a voter V_i and wants them to vote v_0 , but the voter wants to vote v_1 . The adversary must decide which one their coerced

voter voted for in the end. To avoid the trivial case where there would be zero votes for v_0 without the coerced voter, we introduced a balancing voter. Also we add that \mathcal{A} can coerce everyone or no one.

12.1 Paillier

Coercion Game We will now define a game for coercion, for use as the basis of our security proof. In it, the Simulator plays all participants, whilst the Adversary sees everything that is written to the bulletin board.

1. Simulator runs key generation. Flips a coin b .
2. Simulator runs ballot generation, then ballot verification.
3. Adversary specifies votes $(v_{i,0}, v_{i,1})$ for $i = 1, 2, \dots, N$, with $\sum L^{v_{i,0}} = \sum L^{v_{i,1}}$.
4. Simulator has every voter mark their ballot with v_i, b and submits it.
5. Adversary learns which voter submitted which ballot on the bulletin board.
6. Simulator runs counting.
7. Adversary guesses b' .

We represent this game in Figure 14 Now let E represent the event that $b = b'$. We aim to prove, for coercion-resistance, that:

$$2Adv_{\mathcal{A}} = |Pr[E] - \frac{1}{2}|$$

This is a similar game to previous proofs, demonstrated in Figure 14. The difference is the extra information the adversary is able to see, specifically which ballot belongs to which voter.

Theorem 12.1. *Paillier Prêt à Voter has coercion-resistance*

Proof. We can use the same games as in the security proof for privacy, except with the addition that the adversary is able to see which voter submitted which ballot on the bulletin board. We call these versions of the games $G'_0, G'_1, G'_2, G'_3, G'_4$. As this is the only difference, and every game has changed in exactly the same way, games 0 to 3 remain indistinguishable such that $Pr[E'_0] = Pr[E'_1] = Pr[E'_2] = Pr[E'_3]$.

We need to examine G'_4 slightly closer as the ciphertexts are created differently. Since in this game the ciphertexts are random encryptions, they contain no information about anything as they are independent to the permutation π_i and to the intended vote m_i . Therefore, since this ciphertext is the only extra information that the adversary can see in this game, the advantage remains the same as in the privacy proof, i.e. $Pr[E'_4] = Pr[E_4] = \frac{1}{2}$. So since the conditions

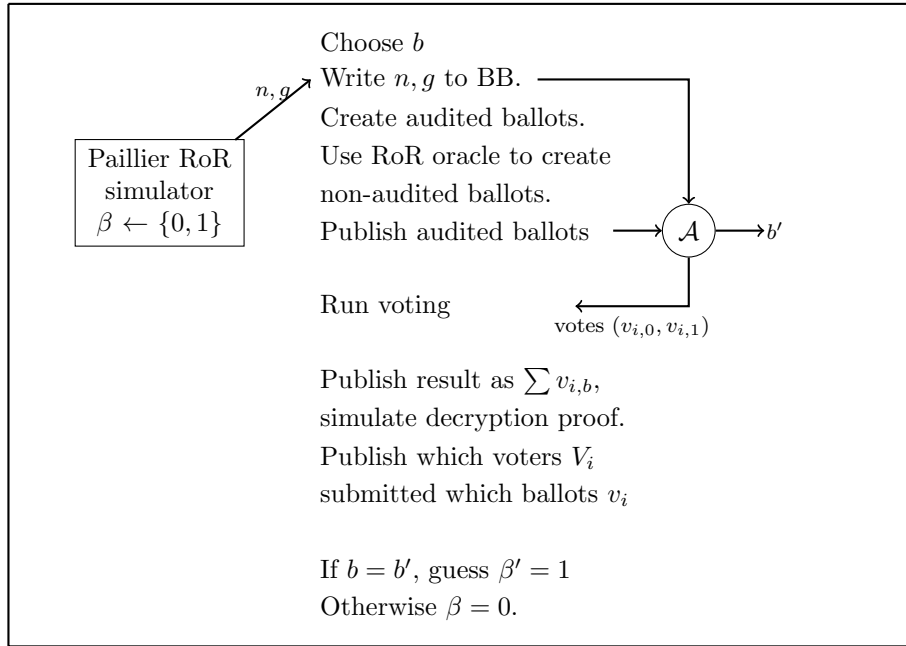


Figure 14: Coercion RoR, Adversary \mathcal{B}

for the security proof of privacy still hold, we can do the calculations as before and find that

$$2Adv_{\mathcal{A}} = |Pr[E] - \frac{1}{2}|,$$

as required. \square

12.2 Mixnet

The proof for Mixnet coercion resistance is very similar, so we will not include it in this paper.

13 Verification Proof

Using the definition of a verifiable system given at the beginning of this section, we apply it to the scenario of our Prêt à Voter protocols, and then prove that they are indeed verifiable.

13.1 Paillier

Suppose that k voters verify the bulletin board, i.e. all the proofs are correct and they can see their right-hand-sides published on the bulletin board.

Say the voters submitted ballots m_1, m_2, \dots, m_k . Then, it is verifiable if the results posted by the bulletin board satisfy

$$result \geq \sum L_i^m$$

for $i = 1, \dots, k$.

Theorem 13.1. *Paillier Prêt à Voter is verifiable.*

Proof. Since we have verified the ballots with cut and choose, we know that the ciphertext corresponding to m_i on the i 'th voter's RHS decrypts successfully to L^{m_i} .

Similarly, from cut and choose, we know that every ciphertext from a ballot decrypts to L^j , $0 \leq j < R$. Therefore, if $\mu_i = \mathcal{D}(c_i)$, then $\mathcal{D}(\pi c_i) = \sum \mu_i$, considered over the integers. As the voters have verified the RHS of their ballots were posted, this confirms that their ciphertexts were among the c_1, c_2, \dots, c_N submitted ciphertexts. The claim follows as, if we re-label the remaining $N - i$ ciphertexts such that they are m_j for $j = i + 1, \dots, N$, we have

$$result = \sum L_i^m + \sum L_j^m$$

□

13.2 Mixnet

The proof for verification for Mixnet is basically the same, as the Mixnet variation also uses Cut-and-Choose, so we will not include it in this paper.

14 Remaining Attacks

In this section we suggest some remaining attacks which we have not addressed, and cannot be resolved in the particular security model we have defined, highlighting the limitations of it. Defense against these attacks for the *Prêt à Voter* protocol is a possible subject for a future paper, and will require a more in-depth security model.

Randomization attack: In this context, since our candidate order is randomised, a randomization attack could be a version of coercion in which the adversary demands the voter marks the top box of the ballot, hence randomising their vote, and this is easy for the adversary to verify the voter complied, since the right hand side is posted on the bulletin board.

A method of defending against this attack is to ensure that the voters can choose from a selection of ballots, so that if they are forced to mark the top box, they can simply choose a ballot which has their preferred candidate at the top. This requires further analysis to solve however, and does not fit into our security model.

Chain Voting: This attack involves the adversary obtaining a blank ballot, possibly by coercing a voter to retrieve one, which the adversary then marks with their chosen candidate. Then they coerce a voter to sneak the pre-marked ballot in, obtain a new blank ballot, then vote with the pre-marked ballot. The voter then sneaks the blank ballot out and gives it to the adversary, with which the adversary can continue coercing voters, creating a chain of falsified votes. [8]

One method that could be discussed to avoid this is adding another player. Known as a Trusted Third Party, their job is to facilitate the voter and either do the marking for them, or observe them marking the ballot in the booth.

Corrupted players In our security proof we have assumed that the players are honest, but it is worth considering the changes in security if one or more of them is corrupted. For example, if the Mixnet servers are corrupt, how would this affect the security proofs? More careful analysis is required to examine this situation, and likewise for versions where different players are cheating.

15 Concluding Remarks

In conclusion, *Prêt à Voter* is a relatively new voter-verifiable voting system, with a focus on transparency of the process and privacy for the voters. In this modern age where our democratic institutions are becoming less trusted, it is an important time to rethink the underlying protocols involved in our voting systems and how we can make them universally fair and more secure.

There are a multitude of options to choose between within the *Prêt à Voter* protocol, and we have discussed some of them here. The basic concept of a randomised candidate order ensuring that the submitted ballot does not reveal to an adversary how each voter voted is an important concept that can be built upon in various ways, depending on the needs of the specific election.

The variations we went into in detail used Paillier encryption and Mixnets respectively, and we provided security proofs to show that they ensured privacy, coercion-resistance and verifiability to the voters. We have only scratched the surface on this topic though, as there is a lot more to consider.

For future papers, there are several remaining attacks to examine for these variations, including those listed. In addition, there are many more potential variations of *Prêt à Voter* to develop and analyse. Finally, more practical implementations could be carried out in real or test elections, to ensure the voting system works just as well in practice as it does in theory.

References

- [1] Peter Y. A. Ryan, Steve Schneider, Vanessa Teague
Chapter 12: Prêt à Voter - the Evolution of the Species, from Real-World

Electronic Voting: Design, Analysis and Deployment
Auerbach Publications, December 2016

- [2] Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, Zhe Xia.
Prêt à Voter: a Voter-Verifiable Voting System from IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL. 4, NO. 4.
IEEE Signal Processing Society, December 2009
- [3] Ben Adida, Ronald Rivest
Scratch & Vote: Self-Contained Paper-Based Cryptographic Voting
Proceedings of the 2006 ACM Workshop on Privacy in the Electronic Society, WPES 2006
- [4] Chris Culnane, Peter Y. A. Ryan, Steve Schneider, Vanessa Teague
vVote: a Verifiable Voting System
ACM Transactions on Information and System Security (TISSEC), 2015
- [5] Pascal Paillier
(1999) *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes.*
Stern J. (eds) *Advances in Cryptology EUROCRYPT 99.* EUROCRYPT 1999. Lecture Notes in Computer Science, vol 1592. Springer, Berlin, Heidelberg, 1999
- [6] Jonathan Katz, Yehuda Lindell
Introduction to Modern Cryptography: Principles and Protocols
Chapman & Hall/CRC, 2007
- [7] Ari Juels, Dario Catalano, Markus Jakobsson
Coercion-Resistant Electronic Elections
RSA Laboratories, CNRS-Ecole Normale Supérieure, Indiana University School of Informatics
- [8] Bo Lipari
Proper Use of Tear-Off Ballot Stubs to Defeat ‘Chain Voting’ Schemes
New Yorkers for Verified Voting
- [9] Ivan Damgård, Mads Jurik, Jesper Buus Nielsen
A Generalization of Paillier’s Public-Key System with Applications to Electronic Voting
Aarhus University, Dept. of Computer Science
- [10] Nina Pettersen
Applications of Paillier’s Cryptosystem
NTNU, Department of Mathematical Sciences

- [11] Stephanie Bayer, Jens Groth
Efficient Zero-Knowledge Argument for Correctness of a Shuffle
University College London

