Theoretical Attacks on E2E Voting Systems*

Peter Hyun-Jeen Lee
Paysafe Group
peter.hyunjeen.lee@gmail.com

Siamak F. Shahandashti
Newcastle University, UK
siamak.shahandashti@ncl.ac.uk

May 5, 2016

Abstract

We give a survey of existing attacks against end-to-end verifiable voting systems in the academic literature. We discuss attacks on the integrity of the election, attacks on the privacy of voters, and attacks aiming at coercion of voters. For each attack, we give a brief overview of the voting system and a short description of the attack and its consequences.

1 Introduction

An election, whether it is paper-based (as in traditional elections) or computer-based (e.g., e-voting), has always been open to threats due to the high stakes for the winner. A survey shows there have been hundreds of election frauds over the last decade in the US alone [1]. Common attacks included vote buying, double voting, stealing absentee ballots and so on which can influence the election outcome. A natural question that arises is then, "how do we ensure the election outcome is correct and if not, detect it?".

In traditional paper-based elections, voters must rely on the trusted party to count the votes. Thus, the verifiability is largely dependant on the trusted party to perform the tallying process correctly and there is no convenient way for voters to independently verify the result. E2E verifiable voting systems on the other hand, try to minimize such dependency and provide every voter means to verify the correctness of the election outcome (i.e., integrity of the election).

E2E verifiable voting systems achieve the verifiability of elections via cast-as-intended (CAI), recorded-as-cast (RAC) and tallied-as-recorded (TAR) guarantees [16]. CAI ensures the voter can verify that his choice is correctly marked on his ballot, RAC ensures the voting system stores the ballot correctly and TAR ensures all the cast votes are correctly included in the final tally. Thus, if malicious voters (or even corrupted election officials) attempt to alter the election result, such attempts will leave evidence which honest voters can use to verify that the integrity of the election is lost.

Privacy is another important property for E2E verifiable voting systems. In the absence of privacy, the relation between the votes and the voters will be visible to others. Such relation then can be used to coerce voters, leading the voters to vote for the choice of the coercer instead of their own. Although such votes will syntactically remain valid, they should be treated as being semantically invalid since they do not

*This is a personally archived version of a chapter by the same title [21] contributed to the book "Real-World Electronic Voting: Design, Analysis and Deployment", Feng Hao and Peter Y. A. Ryan (editors), Series in Security, Privacy and Trust, CRC Press, 2016.

reflect the true intentions of the voters. However, it seems almost impossible (if not impossible) to determine a voter's true intention from a cast ballot. It is thus best to prevent coercions by ensuring the votes remain secret.

A common approach to realizing E2E verifiability is by introducing receipts in the system. These receipts have two main requirements. One is to ensure E2E verifiability of the system by serving as voter verifiable evidence. Another is to do so without revealing any information about how voters have voted. These seemingly contradicting requirements contribute towards making E2E verifiable voting systems complex.

The aim of this chapter is to aid the readers to become aware of various pitfalls while designing an E2E verifiable voting system. In the following, we will review the existing attacks against E2E voting systems in the academic literature. The attacks are divided into three categories - integrity, privacy and coercion, depending on the specific property each attack aims to compromise.

2 Integrity

Integrity ensures the correctness of the election outcome and hence perhaps is the most important property an E2E voting system should possess. This section discusses various attacks against well known E2E verifiable voting systems which aim at compromising the integrity of an election.

2.1 Misprinted ballots attack

In a voting environment, it is generally expected that there are trusted parties (e.g., election officials) who handle critical operations such as verifying identity of a voter, counting the tally and so on. When such trusted parties behave maliciously, one can no longer have confidence in the outcome of an election. Kelsey et al. [17] introduced an attack where the election official intentionally misprints a portion of ballots to alter the election outcome, hence the attack is named misprinted ballots attack. Prior to demonstrating the attack, we start by briefly describing Punchscan in order to familiarize the readers with the system.

Punchscan is an E2E verifiable voting system which makes use of optical scanning of marked ballots. In Punchscan, a ballot consists of two sheets. The front sheet has a list of candidates with associated letters, a serial number and a hole for each candidate. The back sheet has the letters associated with the candidates printed such that when two sheets are overlayed, the letters appear under the holes. A voter casts his vote by marking the letter corresponding to the candidate of his choice with a bingo dauber. He then must choose which sheet (front or back) to keep as a receipt and discards the other one. The receipt can later be checked against the Public Bulletin Board (PBB) released by the election authority.

The attack involves an interaction between a voter and the corrupted election official (the attacker). To begin with, the attacker replaces a portion of ballots with tampered ballots. Then, whenever a voter chooses the front sheet as his receipt, the attacker gives a tampered ballot to the voter. A tampered ballot has two sheets, a tampered back sheet and an untampered front sheet. The back sheet is tampered such that the ordering of letters is swapped. Thus, a voter who intends to cast a vote for the candidate of his choice will instead end up casting a vote for a swapped candidate. An example of misprinted ballots attack is shown in Figure 1. A voter who intends to vote for Jake sees the tampered ballot and marks the second hole then scans the front sheet. The system however, correlates this scanned copy to the

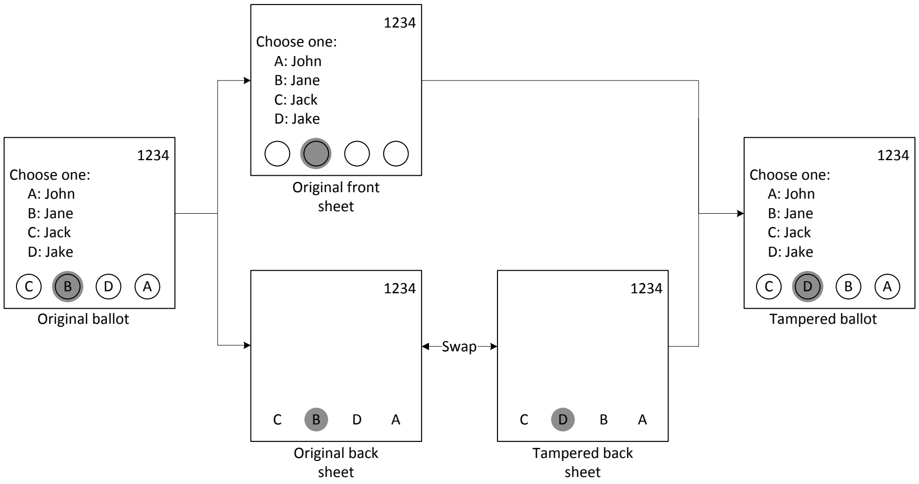


Figure 1: Misprinted ballots attack against Punchscan: Tampered ballot cast by a voter gets linked to the original ballot which results in a vote swap

original ballot and the vote goes to Jane instead. The back sheet which is the only evidence of tampering gets destroyed as a part of the legitimate voting process.

A proposed remedy for this attack is to have the election official commit to a ballot *before* asking the voter to choose a receipt sheet. This way, the election official has a chance of being caught as he no longer has a priori knowledge of whether the voter will keep the front sheet as the receipt or not. For example, if the election official picks a tampered ballot but the voter decides to choose the back sheet as his receipt, then this will allow the voter to later find out that the election official misbehaved.

2.2 Trash Attack

One of the important assumptions in many E2E verifiable voting systems is that the voters should check their receipts. Some voters however choose to discard their receipts to the bin which can be an indication that they will not check their receipts against the PBB. An attacker (e.g., election officials) can exploit this voter behaviour to manipulate the votes corresponding to the discarded receipts with a high confidence of not being caught. This attack is called trash attack [6].

A suggested countermeasure in [6] is to use a hash chain. That is, in order to generate a receipt for a voter, the system creates a hash which is dependent on the current vote as well as the previous vote. However, this approach raises two concerns which do not seem to have been documented in the literature. One concern is user privacy issue. Because each vote is now dependent on another vote, there is an inherent ordering that is present in the receipts. Thus, a receipt can act as publicly verifiable evidence that shows a voter was present at the voting booth for a given period of time. This may discourage privacy-keen voters to participate in elections.

Another concern is performance issue. Helios for example, is a distributed online voting system where multiple voters can simultaneously cast their votes in real time. However, hash chaining being a linear process prohibits concurrent processing of ballots thus a voter may have to wait a considerable amount of time until his vote gets processed.

2.3 Clash Attack

Clash attack [20] is another example of exploiting the weakness in the receipt management of E2E verifiable voting systems. In a sense, clash attack can be viewed as a stronger attack as it works even when the voters verify their receipts against the PBB. The main idea behind clash attack is that the attacker has the power to issue *duplicate* receipts to multiple voters as if they were legitimate receipts. This can be done either by the corrupted authority or by an external attacker who takes control of the voting machine and the PBB. Then, for each duplicated receipt one vote can be safely manipulated. Certain implementations of several well known systems such as Wombat [2], ThreeBallot [27] and Helios [3] were shown to be vulnerable to clash attack. The following sequence of events describes the attack procedure for a variant of Helios.

1. The browser and the PBB are compromised.

2. The browser always uses the same sequence of random numbers $(r_i)_{i=1}^{n}$ to generate i-th ciphertext $C_i = Enc_{pk}(c, r_i)$ where c is the candidate chosen by the voter and n is the maximum number of audits.

3. All the voters who have voted for the same candidate and audited the same number of times receive the same receipt.

4. The attacker keeps one receipt for verification purpose and replaces all the remaining cast ballots then publishes the corresponding receipts on the PBB.

5. Each voter verifies his receipt appears on the PBB and believes his vote was cast correctly.

Clash attack works because the system can be manipulated to produce duplicate receipts without voters detecting it. Thus, even a careful victim who checks his receipt against the PBB will not be able to detect the attack as his receipt will appear on the PBB. This attack can be mitigated if an additional requirement is added to ensure every receipt is unique. One suggested approach is to use the voter contributed randomness in addition to the machine chosen randomness during encryption. Then, assuming the entropy of the voter chosen randomness is high enough (i.e., the randomness chosen by a voter is highly likely to be different from the randomness chosen by another voter), the resulting receipt will be unique.

2.4 Flawed Mix-net

Prior to describing the attacks on mix-nets, a brief introduction to the concept is presented for the readers who are not familiar with mix-nets. Mix-nets were originally proposed by Chaum in 1981 [10]. A mix-net is a series of servers (also known as mixes), each taking a tuple as input and producing a tuple of the same size as output, where each server obfuscates the relation between its input and output by mixing (i.e., applying a random permutation). The aim is to shuffle a tuple securely, that is, the correspondence between the input values to the first mix and the output values of the last mix remains secret even if some mixes are corrupted by an adversary. Figure 2 shows an example of a mix-net.

There are two main types of mix-nets: Chaumian mix-nets [10] and homomorphic mix-nets [23]. In both mix-nets, the input to the first mix-net is a tuple of

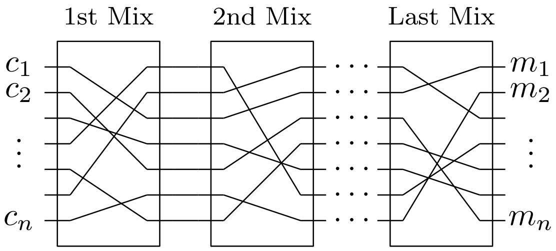


Figure 2: A Chaumian mix-net

ciphertexts. In Chaumian mix-nets, each input ciphertext is calculated by encrypting the plaintext consecutively under the keys of the mixes starting with that of the last mix. During the mixing operation, each mix "peels off" a layer of encryption by decrypting its input values with the last mix peeling off the last layer and outputting the original plaintexts. Figure 2 shows a Chaumian mix-net. In homomorphic mix-nets, each input ciphertext is a homomorphic encryption of the plaintext. During the mixing operation, each mix simply re-randomizes each encryption, resulting in a new ciphertext corresponding to the same plaintext. Eventually, the output tuple of the last mix need to be decrypted to retrieve the original plaintexts. A homomorphic mix-net may be represented similarly to Figure 2 with the only difference that the output of the last mix will be re-encrypted ciphertexts rather than plaintexts.

In order to guarantee the correctness of the mixing, Random Partial Checking (RPC) is proposed [15]. RPC is a well known technique which works by each mix revealing the relation between randomly chosen half of its input and output. In Chaumian mix-nets, this relation is revealed by providing the randomness necessary to encrypt a given output to a given input. In homomorphic mix-nets, it is revealed by exposing the randomness used by the mix-server to re-encrypt. Revealing half of the relations ensures cheating mixes are caught with a high probability, while at the same time the end-to-end input and output relationship remains secret with a high probability thus protecting the sender privacy. To achieve this, mixes are grouped in pairs of consecutive servers, then within each pair, the output ciphertexts of the first mix (which are the same as the input ciphertexts to the second mix) are randomly divided into two sets, and each mix is challenged to reveal the relations for one of the sets.

However, it was shown that the original description of RPC has weaknesses which enable attackers to break the sender privacy and the correctness of shuffling [18]. The attacks work on both Chaumian and homomorphic mix-nets. These attacks are then extended to the RPC implementations of systems such as Civitas [11] and Scantegrity [8].

Khazaei and Wikström [18] observed that in the existing implementations of mix-nets at the time, two crucial well-formedness checks were often missing: checking for duplicate ciphertexts and checking for permutation consistency.

Assume that there are duplicate ciphertexts in the input of a mix in a Chaumian mix-net. They should result in duplicate plaintexts. However, a corrupted mix can replace one of the duplicate outputs with an arbitrary value, while keeping the other one unchanged. If the cheating mix is challenged on any of the duplicate input ciphertexts, it will be able to successfully claim that the input ciphertext corresponds to the unchanged output. The mix is not caught unless it is asked to show the relation for the injected output.

Khazaei and Wikström further observed that in the RPC implementations after a mix was challenged on revealing the relations for some input or output ciphertexts,

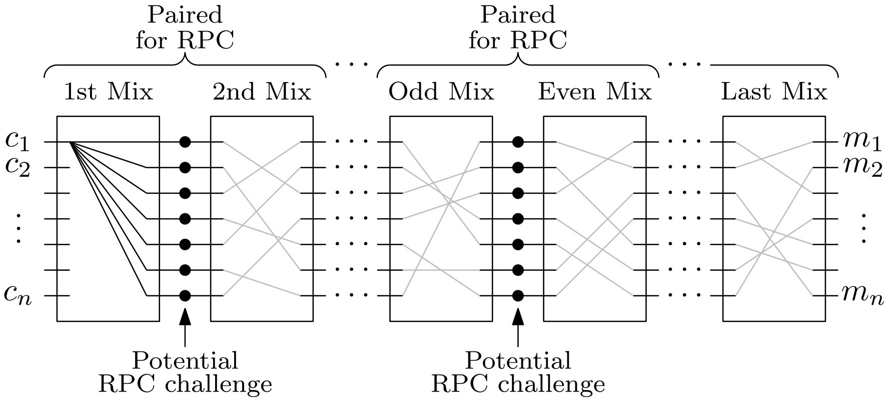


Figure 3: Rigging an election

the only verification test which was carried out was to check if each relation is valid. They argue that this leaves the mix free to claim relations with collisions, e.g., two or more output ciphertexts corresponding to the same input. Lack of such permutation consistency checks enables mixes to get away with incorrect shuffling.

As an example, the attack which breaks the integrity of the election is described below. The rest of the attacks which break the privacy of the election are described in Section 3.

Rigging an Election without detection This attack exploits the lack of permutation consistency checks in RPC. It applies to all homomorphic mix-nets, and to those Chaumian mix-nets that do not implement duplicate ciphertext removal. Duplicate ciphertext removal refers to the practice of inspecting the inputs to a mix and removing any duplicate ciphertexts. Suppose an attacker takes full control of the first mix. Then, he can set multiple output ciphertexts to ciphertexts corresponding to the same input. This will amplify the vote corresponding to the repeated ciphertext and compromise the integrity of the election. An extreme case of this attack where the first mix replaces all the outputs with ciphertexts corresponding to the first input is shown in Figure 3. Note that in RPC the first mix is challenged to reveal relations for half of its *output* only since the first mix is paired with the second mix. Hence the first mix will be able to provide a valid relation for any challenged output and if the relations are not checked for permutation consistency the attack will not be caught. Note that the first mix is not able to provide a valid relation for most of its input ciphertexts, but these are not challenged in RPC. Also note that in general any mix with an odd index can carry out this attack by repeating its input ciphertexts.

The countermeasure proposed by Khazaei and Wikström is to verify permutation consistency in RPC, i.e., to check that all the revealed input-output relations have distinct positions. As they also mention though, this check was missed in the Civitas and Scantegrity implementations of RPC.

3 Privacy

It is important that an E2E verifiable voting system hides how a voter has voted from other voters (ideally even from the system iteslf) thereby providing privacy. While privacy is important in its own right, compromise of privacy can lead to coercion attacks which then can lead to the loss of integrity. In this section, attacks which aim to compromise privacy are discussed.

3.1 Replay Attack

Suppose a small mock election of three voters is run. Two voters Alice and Bob cast their votes legitimately. Then, Charlie duplicates Alice's vote. If both Alice and Bob voted for the same candidate then obviously there is no privacy (i.e., the tally will reveal the winner whom both Alice and Bob have voted for). Had they voted for different candidates, the candidate that Alice voted for will win the election with two votes and hence revealing her vote which in turn will reveal Bob's also. This demonstrates simply *replaying* another voter's vote can result in a breach of privacy. This replay attack was shown to be applicable to Helios 2.0 [12].

Apparently, the aforementioned scenario is somewhat artificial and does not reflect real world elections. It was however, shown that replay attack can be applied to real world elections through a French election case study [12]. The case study was conducted based on a statistical model which reasonably resembles an actual election indicating that the attack does pose a threat in the real world.

As a simple remedy for replay attack, ballot weeding is proposed [12]. Ballot weeding involves checking for duplicate ballots in the system under the assumption the signature of knowledge, which acts as a proof that the cast ballot is well formed, is not malleable (i.e., cannot be altered in such a way that verification still succeeds).

Replay attack can be extended to ballot-blinding attack where each copy of a vote made can be distinct. The technique requires a basic understanding of a well known encryption algorithm called ElGamal encryption. Let $p = bq + 1$ be a large prime where q is also a prime and $b \geq 2$. A generator $g \in \mathbb{Z}_p^*$ with order q is chosen. Further, the secret key $x \in \mathbb{Z}_q$ is chosen and the corresponding public key $X = g^x \pmod{p}$ is computed. Encryption requires choosing $r \in \mathbb{Z}_q$ and computing the ciphertext $(R, S) = (g^r, mX^r)$. (R, S) for a given message $m < p$. (R, S) can be decrypted by computing $m = R^{-x}S$.

Algorithm 1 Blind a ballot (R, S) with a public key X

```
procedure Blind($(R, S)$, $X$))
    Pick a random $z \in \mathbb{Z}_q$
    Compute $(R', S') = (g^z R, X^z S) = (g^{z+r}, X^{z+r} m)$
    return $(R', S')$
end procedure
```

Ballot-blinding attack is similar to the replay attack in that it exploits malleability of the ciphertext to tamper the ciphertext while keeping the vote *unchanged*. The corresponding proof, which is a non-interactive zero knowledge proof for proving well-formedness of the ciphertext, can also be blinded in cooperation with the original voter (i.e., the original voter *consents* another voter to copy his vote *without* revealing his vote to the copier). Thus, allowing the copier to create an undetectable copy of the original vote. As shown in Algorithm 1, the attack requires a randomness z chosen at the time of blinding with the public information, the ciphertext and the public key. Blinding the accompanied proof is more technically involved and interested readers are referred to [13] for details.

Desmedt and Chaidos [13] note that this attack works for on-line voting but not for the polling station-based voting due to the cooperation requirement. The main difference between the two types of voting systems is that it is relatively easier to enforce procedural requirements during the election in polling station-based voting than in on-line voting.

3.2 Kleptographic Attack

Algorithm 2 Embed a secret message c into the randomness part of an ElGamal signature (r, s)

```
procedure EMBED($c$)
    $p = qm + 1$ ▷ $m$ is smooth and $\mathbb{F}_p^*$ is a subgroup of order $q$ generated by $g^m$
    $k' \in_R \mathbb{Z}_q$
    $k = c + k'm \pmod{p-1}$
    return $r = g^k \pmod{p}$
end procedure
```

Algorithm 3 Retrieve the secret message c given an ElGamal signature (r, s)

```
1: procedure RETRIEVE($r$)
2:     find $z$ such that $(g^q)^z = r^q \pmod{p}$                ▷ i.e., $z = \log_{g^q} r^q$
3:     $c = z \pmod{m}$
4:     return $c$
5: end procedure
```

E-voting schemes which make use of *randomness* are potentially vulnerable to kleptographic attack. For example, Helios [4] uses ElGamal encryption as the encryption method which is a randomized algorithm. A randomized algorithm takes a stream of random bits in addition to the input (e.g., vote) to produce an output. Thus, each encryption of the same vote will look distinct which helps to achieve the voter privacy in elections.

Kleptographic attack works by constructing a subliminal channel in the random bits used in encryptions. The presence of a subliminal channel does not change the protocol nor can be detected without reverse engineering the software. This makes a subliminal channel an attractive choice for leaking confidential information to a third party.

The attack was initially motivated by the question whether there exists a signature scheme with a broadband covert channel without revealing the sender's private key. This was shown to be true using ElGamal signature as an example [5]. Algorithms 2 and 3 show how an attacker can embed a secret message and retrieve it, respectively. Note that step 2 in Algorithm 3 is computable because the group order is smooth (e.g., let B be the smoothness bound, that is for a prime $p = qm+1$, B is the largest prime in m, it is computable in time $O(\sqrt{B})$ using Pohlig-Hellman [25] and Pollard's rho [26]).

A typical high level structure of ElGamal signature can be viewed as a tuple (r, s) where s is the signature on a message and r is the randomness used in creating s. The similar structure is used for ElGamal encryption and hence the attack is immediately applicable. The authors further show how to convert their covert broadcast channel into narrowcast by pre-sharing secrets among the communicating parties.

It was later shown that such a channel also exists in DSA-like schemes [28]. Gogolewski et al. [14] demonstrated the kleptographic attacks against a number of e-voting schemes [19, 7, 22, 9]. Kleptographic attack poses a real threat to e-voting schemes using randomness as it is hard to detect. It should be noted however that kleptographic attack is tailored to exposing secrets. Thus, provided that the publicized data remain unmodifiable (e.g., encrypted ballots stored on append-only PBB), the integrity of election result may still be retained.

Gogolewski et al. [14] suggest a potential approach to solving this problem by pre-generating all the randomness and use these as random-tapes. Then, zero knowledge proofs can be used to show that the generated ballots indeed used the randomness from the random-tapes. However, the authors note that zero knowledge proof itself makes use of randomness which again can be a source for a kleptographic channel therefore this does not completely solve the problem.

3.3 Pfitzmann's attack

Pfitzmann's attack [24] can be adopted to compromise the sender privacy in a homomorphic mix-net with RPC. Suppose the first mix is corrupted. The attacker targets an input ciphertext c sent by a sender S to the first mix. The attacker then chooses a random δ and, while keeping c, replaces another output of the first mix with $c^* = c^\delta$. Finally, the attacker searches for a pair of plaintexts m and m^* output by the last mix such that $m^* = m^\delta$. This indicates to the attacker that with all but negligible probability, m and m^* respectively correspond to c and c^*. This leads to the conclusion that m was sent by S, thus breaking sender privacy for S. RPC will detect this attack with a probability of $\frac{1}{2}$ since only half of the input-output relations for the first mix are challenged.

Pftitzmann's attack can be generalized to compromise the privacy of s senders while keeping the probability of detection at $\frac{1}{2}$ [18]. Now the attacker targets ciphertexts $c_1, ..., c_s$ and chooses $\delta_1, ..., \delta_s$. Then, the first mix replaces one of its outputs with $c^* = \prod_{i=1}^{s} c_i^{\delta_i}$. By observing the outputs of the last mix, the attacker can identify $s+1$ messages $m_1, ..., m_s, m^*$ respectively resulting from $s+1$ ciphertexts $c_1, ..., c_s, c^*$ if $m^* = \prod_{i=1}^{s} m_i^{\delta_i}$ satisfies. If so, the attacker concludes that c_i is an encryption of m_i for every i, breaking sender privacy for s senders. The probability that the attack is discovered by RPC is still $\frac{1}{2}$ since only one of the outputs of the first mix are replaced.

3.4 Duplicate ciphertext attack

Duplicate ciphertext attack as its name suggests works by injecting multiple sets of duplicate ciphertexts into the mix-net [18]. Then, a special relation imposed among the ciphertexts enables the attacker to determine the corresponding plaintexts. The attack is applicable to the Chaumian mix-net with RPC. Together with the assumption that the attacker corrupts $s(s+1)/2$ senders and the first and last mixes, the attack works as follows:

1. The attacker targets the first s ciphertexts input to the first mix: c_i for $1 \leq i \leq s$.

2. The attacker obtains c'_i by decrypting the outer-most encryption layer of the s ciphertexts with the secret key of the first mix.

3. The attacker makes i independent encryptions of c'_i (i.e., 1 encryption for c'_1, 2 encryptions for c'_2, and so on) using the public key of the first server.

4. The previous step generates $1 + 2 + ... + s = s(s+1)/2$ ciphertexts and the attacker sends each of these to a corrupted sender to submit.

5. For an ith targeted ciphertext there are i duplicates. Thus, the attacker can identify the correspondence between the targeted ciphertexts and the input ciphertexts to the last mix based on the number of duplicates that exists in the input list of the last mix. Hence, the attacker breaks the privacy of the s targeted ciphertexts.

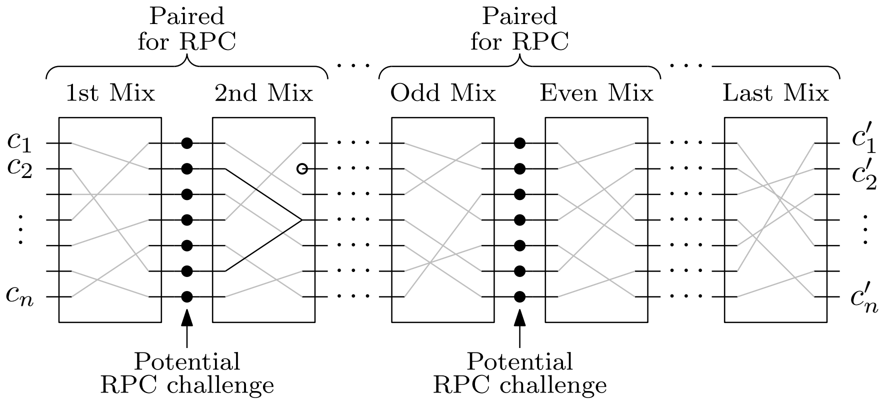


Figure 4: Breaking privacy without detection

The attack works because there is no check for duplicates at every mix. Note that RPC only applies duplicate removal at the input of the first mix and this attack is able to bypass that. A simply remedy for this attack therefore is to mandate the check for duplicates at every mix. This attack can compromise the privacy of $\mathcal{O}(\sqrt{N})$ senders as $s + s(s+1)/2$ cannot be greater than the maximum number of senders N.

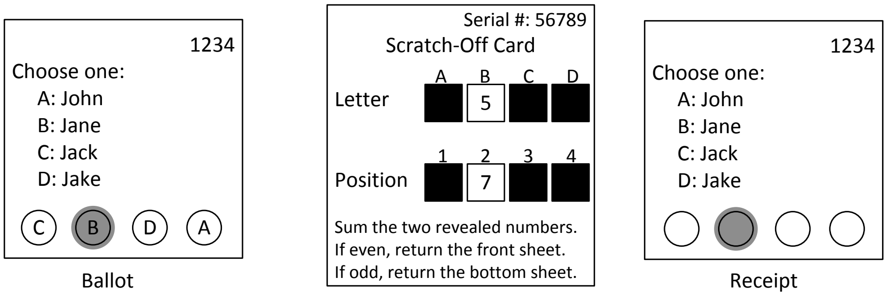
3.5 Breaking privacy without detection

This attack, proposed in [18], takes advantage of lack of permutation consistency checks in RPC to compromise the privacy of the votes. Here, the adversary needs to corrupt two input ciphertexts to the system and the first two mix servers. The attack works only for homomorphic mix-nets.

Assume the adversary can control the first two ciphertexts c_1 and c_2. We describe how the attack works based on Figure 4. The adversary chooses c_1 and c_2 to be re-encryptions of the same ciphertext. The first mix shuffles honestly, but it keeps records of the positions of c_1 and c_2 in its output and the randomization factors used for those two. The second mix is now able to assign both of these input ciphertexts to one output ciphertext since all the randomization factors between the two are known. This enables the second mix to have one of its output locations (second location in Figure 4) free to be set as desired. The adversary sets this ciphertext similar to the generalized Pfitzmann's attack equal to a ciphertext calculated from a set of s targeted input ciphertexts. This way the attacker will be able to find the correspondence between the input and output ciphertexts for the s chosen input ciphertexts and hence break the privacy of the voters who cast the votes embedded in the targeted ciphertexts. The proposed countermeasure to this attack is to implement permutation consistency checks in RPC.

4 Coercion

If an e-voting system leaves evidence of how voters have voted (e.g., receipt with a pattern which can be used to deduce the vote cast with a high probability), the system becomes vulnerable to coercion attacks. The vulnerability may be inherent in the system due to design flaw or could have been introduced by external means (e.g., scratch-off cards). This section describes how such coercion attacks can influence the election outcome and how they can be mitigated.



Sum the two revealed numbers.
If even, return the front sheet.
If odd, return the bottom sheet.

Figure 5: Scratch-off card attack from a voter who honestly scratches off the pads

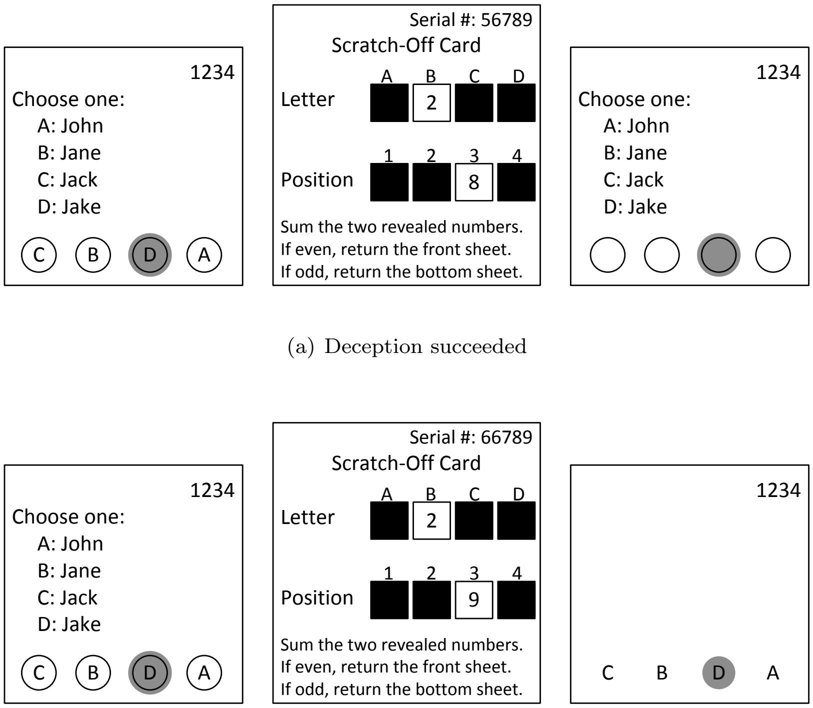
4.1 Forged ballot

In Punchscan and Prêt à Voter, a ballot splits into two halves. While combined halves allow humans to read the vote, each half does not reveal the vote and acts as a receipt which voters can take home. Therefore, Punchscan and Prêt à Voter both require that voters destroy the halves of their ballots. If a voter secretly brings a forged ballot to the polling station, then he can destroy the forged ballot instead of the actual ballot after his vote [17]. This allows the voter to retain the complete proof of his vote leaving the system vulnerable to coercion attacks. A possible solution to this problem is to force the election officials to check the IDs of the two halves. Then, the voter has a risk of getting caught if he is unable to guess in advance the ID of the ballot that he will be given at the polling station.

4.2 Vote against a candidate

Another attack that works against Punchscan causes votes to be randomly distributed among the candidates [17]. Suppose the attacker wants Jake who is a strong candidate to lose the election in Figure 1. The attacker will offer: (1) \$10 for any front sheet receipt showing "D: Jake" with the first hole marked; (2) \$5 for any back receipt marked for "D". If there are n number of vote sellers, each of them will return the front sheet receipt showing "D: Jake" with the first hole (or any single fixed hole would work too) marked if he gets such a ballot. This randomizes the votes away from Jake assuming the letters "A-D" have even chance of occurring at each hole because "D" will appear on the first hole with a probability of $\frac{1}{4}$ only[1]. Returning a back receipt marked for "D" then indicates that the voter did not vote for Jake since he would have been better off with marking the first hole if he indeed had a ballot showing "D: Jake". More concretely, let $\mathcal{E}_1$ be the event that a voter gets a ballot with "D: Jake" and $\mathcal{E}_2$ be the event that "D" appears in the first hole. Then, this will result in about $\frac{1}{n^2}$ (since $\Pr(\mathcal{E}_1) \approx \frac{1}{n}$ and $\Pr(\mathcal{E}_2) \approx \frac{1}{n}$) of voters voting for Jake. Now, the remaining $\frac{n^2-1}{n^2}$ votes will be distributed evenly among $(n-1)$ candidates giving each candidate $\frac{n^2-1}{n^2}/(n-1) = \frac{n+1}{n^2}$ votes.

[1] Indeed, if Jake were to be a weak candidate (e.g., supported by less than 25% of the voters) then this attack will have the opposite effect. However, in such a case Jake would not be a favourable target to an adversary.



(a) Deception succeeded

(b) Deception caught

Figure 6: Scratch-off card attack from a voter who dishonestly scratches off the pads

4.3 Scratch-off card attack

The key idea behind this attack is that *possession of one receipt and knowledge of the other (destroyed) sheet is sufficient to determine the voter's selection* [17]. It is possible to force non-interactive (i.e., does not require vote buyer-seller interaction) challenge and response at the time of voting by using scratch-off cards.

Suppose voters who have pledged to vote for Jane are given scratch-off cards with two rows of pads. As shown in Figure 5, the first row has a pad for each letter that can be associated with Jane and the second row has a pad for each possible position. If a voter honestly scratches the two pads, then he would scratch the pad below the letter B for Jane and the pad below position 2 as the letter B appears on the second hole, which will reveal 5 and 7 respectively. The *challenge* to the voter is the sum of these two numbers and the voter's *response* is to choose which sheet (either front or back) as the receipt. If the sum is even then the voter should return the front sheet, otherwise the back sheet. Figure 6 shows the possibilities if the voter behaves dishonestly.

A suggested countermeasure is to force the voter to choose a receipt sheet before seeing the ballot [17]. Then, the voter has the risk of being caught as he cannot foretell whether the sum will be even or odd, putting the attacker at an advantage. The attacker's power can be weakened if the voter is allowed to spoil his vote, thus allowing him to recast his vote until he obtains a ballot which satisfies the scratch-off card.

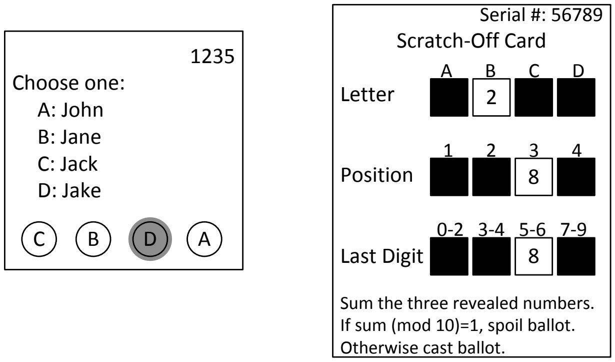


Figure 7: Punchscan: Spoiling ballot

4.4 Spoiling ballots

This attack extends the scratch-off card attack by requiring the voters to commit the serial numbers which appear on their ballots in addition to forcing the voters to commit to their choices. As shown in Figure 7, the scratch-off card now has an additional row of pads. After the voters commit their choices, each voter must compute the sum of the three numbers and if the sum is congruent to 1 (mod 10), he must spoil the ballot. The strength of this attack is that the voter no longer has to depend on the commitment he makes prior to seeing his ballot. He can return any sheet (front or back) as long as the condition to spoil the ballot does not hold although it may have lower probability of catching the voter's deception. For example, assuming the sum of three numbers are evenly distributed the deception by voters will be caught with a probability of $\frac{1}{10}$.

4.5 Pay-Per-Mark & Pay-for-Receipt

The following two attacks show how the presence of patterns in receipts can leave the system open to coercion attacks. These attacks work because the machine randomly marks opscan bubbles which satisfy voters' choices. If voters are given the full control of how to mark opscan bubbles, then the attack can be mitigated.

Pay-Per-Mark. Recall that in ThreeBallot, a voter marks exactly two optical scan bubbles for the candidate he votes for and exactly one for the candidates he does not wish to vote for. Suppose a vote buyer offers some cash reward per mark for a particular political party. Then, a voter who does not vote for the party will end up with a receipt with $\frac{n}{3}$ marks for the party on average where n is the total number of questions. A voter who votes for the party on the other hand will end up with a receipt with $\frac{2n}{3}$ marks on average.

Pay-for-Receipt. This attack requires that a voter gives his receipt to the attacker after he has voted. The attacker then checks the receipt to see whether it matches to one of the expected ballots. Suppose a voter is directed to vote for John but not for Jane as shown in the ThreeBallot in Figure 8. If the voter votes as directed by the attacker then the voter is guaranteed to obtain a valid receipt. If instead the voter votes for Jane then the voter will obtain a valid receipt with a probability of $\frac{1}{3}$. If the voter votes for neither of them, then he will obtain a valid receipt with a probability of $\frac{2}{3}$.

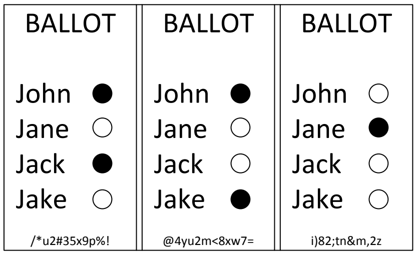


Figure 8: Pay-for-Receipt

5 Conclusion

We have reviewed various attacks against E2E verifiable voting systems in the literature. The attacks exploited vulnerabilities both in the procedural aspect (e.g., not requiring users to check for duplicate receipts) and the technical aspect (e.g., exploiting homomorphic property of underlying encryption). This suggests that in order to build an E2E verifiable voting system, relying on procedural enforcement or technology alone may be insufficient - one must consider the both.

We have also witnessed the attacks which focus on coercion by offering incentives to corrupted voters and other attacks which are aimed at influencing the election result more aggressively by actively tampering votes in cooperation with the corrupted authority. Many of the attacks came with the remedies, which unfortunately however, do not always completely resolve the problems. Thus, providing the complete solutions for these known attacks will be the first step towards convincing the general public to accept the new voting technology as an improvement to the paper-based voting.

6 Acknowledgements

The authors would like to thank Syed Taha Ali, Feng Hao and Shahram Khazaei for reviewing an earlier version of this chapter. This work is supported by ERC Starting Grant No. 306994.

References

[1] RNLA Survey. `http://www.rnla.org/survey.asp`.

[2] Wombat voting system. `http://www.wombat-voting.com`.

[3] Ben Adida. Helios: Web-based open-audit voting. In *USENIX Security Symposium*, volume 17, pages 335–348, 2008.

[4] Ben Adida, Olivier De Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: analysis of real-world use of Helios. In *International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, 2009.

[5] Ross Anderson, Serge Vaudenay, Bart Preneel, and Kaisa Nyberg. The Newton channel. In *Information hiding*, pages 151–156. Springer, 1996.

[6] Josh Benaloh and Eric Lazarus. The trash attack: An attack on verifiable voting systems and a simple mitigation. Technical report, Technical Report MSR-TR-2011-115, Microsoft, 2011.

[7] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE security & privacy*, 2(1):38–47, 2004.

[8] David Chaum, Aleks Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan Sherman, and Poorvi Vora. Scantegrity: End-to-end voter-verifiable optical-scan voting. *Security & Privacy, IEEE*, 6(3):40–46, 2008.

[9] David Chaum, Peter Y A Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In *Computer Security – ESORICS 2005*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139. Springer Berlin Heidelberg, 2005.

[10] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[11] Michael R Clarkson, Stephen Chong, and Andrew C Myers. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354–368. IEEE Computer Society, 2008.

[12] Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.

[13] Yvo Desmedt and Pyrros Chaidos. Applying divertibility to blind ballot copying in the Helios internet voting system. In *Computer Security–ESORICS 2012*, pages 433–450. Springer, 2012.

[14] Marcin Gogolewski, Marek Klonowski, Przemysław Kubiak, Mirosław Kutyłowski, Anna Lauks, and Filip Zagórski. Kleptographic attacks on e-voting schemes. In *Emerging Trends in Information and Communication Security*, pages 494–508. Springer, 2006.

[15] Markus Jakobsson, Ari Juels, and Ronald L Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX security symposium*, pages 339–353, 2002.

[16] Hugo Jonker, Sjouke Mauw, and Jun Pang. Privacy and verifiability in voting systems: Methods, developments and trends. *Computer Science Review*, 10:1–30, 2013.

[17] John Kelsey, Andrew Regenscheid, Tal Moran, and David Chaum. Attacking paper-based E2E voting systems. In *Towards Trustworthy Elections*, pages 370–387. Springer, 2010.

[18] S Khazaei and D Wikström. Randomized partial checking revisited. In *Topics in Cryptology - CT-RSA 2013*, volume 7779 of *Lecture Notes in Computer Science*, pages 115–128. Springer Berlin Heidelberg, 2013.

[19] Marek Klonowski, Mirosław Kutyłowski, Anna Lauks, and Filip Zagórski. A practical voting scheme with receipts. In *Information Security*, pages 490–497. Springer, 2005.

[20] R Kusters, T Truderung, and A Vogt. Clash attacks on the verifiability of e-voting systems. In *2012 IEEE Symposium on Security and Privacy*, pages 395–409, 2012.

[21] Peter Hyun-Jeen Lee and Siamak F. Shahandashti. Theoretical Attacks on E2E Voting Systems. In Feng Hao and Peter Y. A. Ryan, editors, *Real-World Electronic Voting: Design, Analysis and Deployment*, Series in Security, Privacy and Trust. CRC Press, 2016. (in press).

[22] C Andrew Neff. Practical high certainty intent verification for encrypted votes. votehere (2004), 2004. `http://www.votehere.net/vhti/documentation`.

[23] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259. Springer Berlin Heidelberg, 1994.

[24] Birgit Pfitzmann and Andreas Pfitzmann. How to break the direct RSA-implementation of mixes. In *Advances in Cryptology–EUROCRYPT'89*, pages 373–381. Springer, 1990.

[25] Stephen C Pohlig and Martin E Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (corresp.). *Information Theory, IEEE Transactions on*, 24(1):106–110, Jan 1978.

[26] John M Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, Jul 1978.

[27] Ronald L Rivest and Warren D Smith. Three voting protocols: Threeballot, VAV, and Twin. In *USENIX Workshop on Accurate Electronic Voting Technology*, volume 16. USENIX Association, 2007.

[28] Adam Young and Moti Yung. Bandwidth-optimal kleptographic attacks. In *Cryptographic Hardware and Embedded Systems — CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 235–250. Springer Berlin Heidelberg, 2001.