Merging Prêt-à-Voter and PunchScan

Jeroen van de Graaf
Laboratório de Computação Científica
Universidade Federal de Minas Gerais
jvdg@ufmg.br

Abstract

We describe a variation of the Prêt-à-Voter voting protocol that keeps the same ballot layout but borrows and slightly modifies the underlying cryptographic primitives from Punchscan, which is based on bit commitments. By using unconditionally hiding bit commitments, our protocol obtains unconditional privacy. We suggest a way to make cheating on the computational binding bit commitments impossible under assumptions that seem plausible for large-scale elections. Also we show ways to have several races on the Prêt-à-Voter ballot, showing that with respect to ballot layout the protocols are almost identical.

1 Introduction

Over the last few years we have seen a sequence of papers on voter-verifiable elections. The idea of such systems is that the voter takes home a receipt which allows her to verify that her vote is included in the tally without revealing any useful information about her vote. Though this idea is not new[1], Chaum's paper [2] arguably gave a new impetus to this line of research (see also [3]).

This protocol has been improved upon in two significant ways. First there is a protocol called Prêt-à-Voter (PaV), as described in [4]. Subsequently, Chaum developed PunchScan (PS). See the site *www.punchscan.org* for demos and technical descriptions[5, 6].

Both protocols have important advantages over [2], such as a simpler ballot lay-out, and pre-printed ballots on which the voter marks his preferences with a pen, thus insuring that the voting machine (DRE) does not learn the vote. This improvement implies a significant leap forwards, since until then most e-voting systems had the disadvantage that a machine learns how a person voted, and protecting thousands of voting machines is very difficult, apart from leaving the voter uneasy.

There are also some important differences between PaV and PS: First, their ballot layouts are slightly different, and so is the mechanics of voting. But as we will show, these differences are marginal.

Second, PaV uses decryption mixing as the most important primitive, whereas PS uses a Bit Commitment (BC) scheme on a cleverly constructed audit table which is published on a web site, and that uses permutations to hide the links between the voters and the votes cast. This is an important breakthrough: a BC scheme is much simpler to implement and to explain to non-cryptographer than mixing. And from a theoretical point of view it makes a clear link between BC schemes and voting protocols, with interesting implications for unconditional (or everlasting) privacy.

The following list contains a summary of our results and is at the same time the outline of our paper:

Ballot layout We will give a detailed description of the differences in the ballot layouts of Prêt-à-Voter and PunchScan, and show that these differences are almost completely irrelevant from a practical point of view. In fact, our analysis shows that the PaV approach is slightly simpler; PS seems to complicate things without a real advantage. And in [7] it is argued that PS is

superior over PaV with respect to ballot layout, but we show in Apendix A how to make PaV ballots that are virtually identical to the PS ones.

New protocol and proofs We present a simpler protocol by merging PaV and PS as follows: we maintain PaV's (simpler) ballot layout but we borrow the underlying cryptographic primitives from PS. Apart from giving us a thorough understanding of the similarities and differences between the two protocols, the final result seems superior to both because compared to PaV it disposes of mixing, while compared to PS it results in a simpler ballot lay-out. We also present proofs that our protocol is secure. We do this by using the set of requirements for a secure voting system listed in [8].

Voting with unconditional privacy Though the PunchScan papers fail to mention this (they propose to implement the BCs using classical encryptions), it is a trivial observation that by using unconditionally hiding bit commitments, unconditional (or everlasting) privacy can be obtained. This is important since so far in most voting protocols the privacy is only computational. We discuss this topic in detail and suggest a way to make cheating on the computational binding bit commitments impossible under some plausible assumptions.

In addition, Appendix B provides a very brief description of the cryptography of Punchscan.

Though we made an effort for this paper to be self-contained, some familiarity with the general setting and the terminology of voting protocols will be helpful.

Comparison with other work

The underlying cryptography in this paper is strongly inspired on PunchScan, but our description is more general, more formal and we present proofs. Though [5] and [6] provide a detailed protocol description for the case of a Yes/No vote, the two papers seem derived from a software specification, with less emphasis on cryptographic aspects. With respect to PaV a small warning is appropriate since various, rather different versions called Prêt-à-Voter exist (see for instance [9]). We are basing our discussion on [4]. An earlier, four-page version of this paper was presented as a short paper[10].

We have chosen to focus on PaV and PS since we believe they are currently the most promising candidates for voting protocols with end-to-end verifiability. Though the approach of Neff[11] is very interesting, also mathematically, this system has, in our opinion, three serious disadvantages: First, the protocol is too complicated for non-cryptographers. Second, the voter is supposed to verify that a small random string presented on the screen is printed on a receipt, but no protection exists if the machine bluntly lies. And third, the machine learns how the voter votes. And Rivest's Three-Ballot scheme[12] is hurdled with problems, including the leakage of information while the vote is going on[13].

2 Comparing the ballot layouts of PaV and PS

2.1 The Prêt-à-Voter ballot

The ballots used in PaV are described in detail in [4], section 4. A canonical ordering of candidates must be defined. In this section it is 1: Alice, 2: Bob, 3: Charles, 4: Diane. An example ballot looks like this:

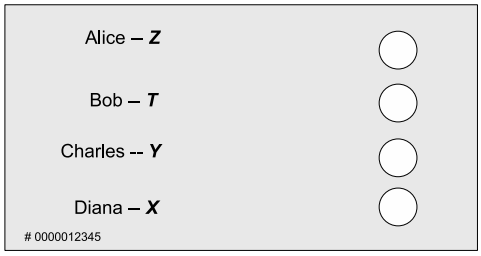
2: Bob	
4: Diana	✗
1: Alice	
3: Charles	
(permutation: 1->3, 2->1 3->4, 4->2)	`Qqkr3c`

The left part contains a permutation of the candidates. The right part is empty except for the last row, and the voter votes by putting an ✗ in one of its first four cells. The magic string `Qqkr3c` (in reality probably longer) is an encryption of x, encrypted with the public keys of the mixes.

Casting the vote consists of separating the left and the right columns, destroying the left column and scanning the right column. Either manually or through OCR the row containing the ✗ and the encryption of the offset are associated to the ballot image. The voter can take the right column home as a receipt.

2.2 The PunchScan ballot

The bottom and the top layer of a typical PS ballot is depicted below. We have used X, Y, Z and T as the auxiliary symbols that make the connection between the option chosen on the top and bottom layers. We could have used any set of 4 different symbols, or four colors etc. Observe that on both layers the symbols are permuted.



When the two layers are superimposed, a vote is cast by using a dauber that affects both layers simultaneously, leaving a mark on both layers. The voter chooses one of the two layers to take with him as a receipt, the other layer is destroyed.

2.3 The differences between PaV and PS

We now list the differences between Prêt-à-Voter and PunchScan.

1. PaV and PS have a different ballot layout: PaV uses one layer that will be cut into a left and a right side, whereas PS uses two different layers.
2. In PaV only the voter options on the left are permuted, while in PS both the voter options and the place to mark the option are permuted.
3. In PaV the mark is placed on the right side which will act as a receipt and the left side will be destroyed whereas in PS a mark is placed on both layers and the voter gets to choose which layer he keeps and which gets destroyed.

4. PaV has the encryption of the permutation printed on the ballot, whereas PS only has a serial number for each ballot which functions as a pointer to other informations that goes through an auditing process.

5. PaV uses decryption mixing as the most important primitive, whereas PS uses a Bit Commitment scheme.

The differences between Prêt-à-Voter and PunchScan with respect to ballot layout can be illustrated explicitly using a ballot with two layers and using arrows. This is shown in Appendix A. There we also show how to have several races on the PaV ballot, which shows that for practical purposes the PaV and PS ballot layouts are virtually equal.

3 The new protocol: Preliminaries

3.1 Ballot layout and how to cast a vote

We will be using the Prêt-à-Voter ballot layout like the one described in Section 2.1, but for simplicity of exposition we restrict ourselves to *cyclic* permutations. We stress, though, that our protocol can be generalized to arbitrary permutations. We use the same canonical ordering but start counting from 0.

3: Diane	✗
0: Alice	
1: Bob	
2: Charles	
(Offset $x = 1$)	#0000012345

As we see, the left part contains a cyclic permutation (shift) of the candidates; in this example the offset is $x = 1$. Obviously x must be different for each ballot.

3.2 The audit table

Since Punchscan uses two permutatons, both on the top and the bottom layer, a straightforward idea is to break the offset value x in two, i.e. to choose x' and x'' random such that $x = x' + x'' (\mathrm{mod}\ m)$. The Election Authority(EA) commits to x, x' and x'' publicly, and use a unique ballot id number n to establish the link between the BCs published and the printed ballot.

We introduce the following notation: x is the offset; y is the number of the row on the ballot marked by the voter, counting from 0 to $m-1$; v is the actual vote, that is, the row chosen with respect to the canonical representation. Obviously, $y = x + v (\mathrm{mod}\ m)$, where the modulus m is the number of candidates on the ballot; in the example $m = 4$.

Let us now describe the audit table to be created by the Election Authority (EA) before the election (which is very similar to Punchscan's). Note the hats on the symbols for some columns; these mean that each cell in that column is a bit commitment. The columns labelled y, $[y - x']$ and v will remain empty until the counting of the votes, as we will see later. At that time the voter ids, n, and the scanned image of the ballot receipts, r, will be published as well.

left (index i)					middle (index j)					right (index k)
i	$\widehat{x}$	n	r	y	$\widehat{\pi_1^{-1}}$	$\widehat{x'}$	$[y-x']$	$\widehat{x''}$	$\widehat{\pi_2}$	v
1										
...										
...										
$2S$										

Observe that the table is divided in a left, middle and right part. Rows corresponding to the same ballot in the left and middle part are permuted according to π_1, in the middle and the right part according to π_2. This is necessary to hide the link between the voter on the left and his vote on the right. To stress this point we use different indexes i, j and k to refer to the different rows of the three parts, and often we will implicitly assume that $j = \pi_1(i)$ and $k = \pi_2(j)$. For instance when we write $y(i) = x(i) + v(k) (\mathrm{mod}\ m)$ we are talking about the values corresponding to the *same* ballot. The columns labeled $\widehat{\pi_1^{-1}}$ and $\widehat{\pi_2}$ are used as commitments for these two permutations. During the audits, BCs are opened to verify that the EA did not cheat, as explained below.

In fact, as will become clear when we discuss the post-election audit, it is necessary to create T copies of the middle and the right part of the audit table, each with the same value x, but with different value π_{1t}^{-1}, x'_t, x''_t and π_{2t}. Here T is a security parameter and t a subscript between 1 and T.

4 The protocol

Phase 1: The pre-election audit

The general idea is as follows: since a ballot must be used either during the election or for auditing, we create and print twice the number of ballots needed and audit half of them. The other half will be used for the election.

Step 1.1 The EA defines election constants such as the names and the number of candidates, m and the names and the total number of voters, S.

Step 1.2 For each $t \in \{1, \ldots, T\}$, the EA chooses 2 random permutations π_{1t} and π_{2t} on the set $\{1, \ldots, 2S\}$. Also, for each $i \in \{1, \ldots, 2S\}$, $j_t = \pi_{1t}(i)$ the EA chooses $x(i)$, $x'_t(j_t)$ and $x''_t(j_t)$ random such that $x(i) = x'_t(j_t) + x''_t(j_t) (\mathrm{mod}\ m)$.

Step 1.3 For each $i \in \{1, \ldots, 2S\}$, $t \in \{1, \ldots, T\}$, the EA commits to $x(i)$, $\pi_{1t}^{-1}(j_t)$, $x'_t(j_t)$, $x''_t(j_t)$ and $\pi_{2t}(j_t)$. For each i and t these commitments represent one ballot, which is printed (where $j_t = \pi_{1t}(i)$).

Step 1.4 The Auditors divide the set of $2S$ ballots randomly in an audit set A and election set E, both of size S.

Step 1.5 The EA now opens all bit commitments related to A: it must open all rows i in the left part of the table if $i \in A$ and all rows with index j_t in the middle part of the table if $j_t = \pi_{1t}(i)$ and $i \in A$.

Step 1.6 The Auditors check that the ballots and the entries in the audit table were created honestly, i.e. that $\pi_{1t}^{-1}(j_t)$ and $\pi_{2t}(j_t)$ are consistent with permutations (no two-to-one mappings), and that $x(i) = x'_t(j_t) + x''_t(j_t) (\mathrm{mod}\ m)$ holds for each row opened, and that the serial number i and the offset $x(i)$ printed on the ballot correspond with the entries in the table.

Step 1.7 All ballots from the audit set A are discarded; the ballots with serial number in E are used for the election.

Phase 2: The election

The voter acts as follows:

Step 2.1 The voter identifies herself using existing mechanisms.

Step 2.2 The voter chooses a ballot $i \in E$ and marks her choice inside the booth by putting an ✗, as already described in Section 2.1.

Step 2.3 She separates the two parts of her ballot and witnesses how the left part is shredded.

Step 2.4 The voter has her receipt (the right side) scanned. Then the row number of her mark, $y(i)$, is interpreted (by OCR or by a poll worker) which superimposes an arrow or other symbol on the scanned image. The voter confirms this interpretation, which correponds to her casting the vote. The voter takes her ballot and a print-out of the scanned image with arrow with her as a receipt. (For an alternative way of capturing the voter's preference, see the last paragraph of Section A.2 in the Appendix).

Phase 3: Publishing the results

Step 3.1 After the election, the EA publishes the voter's id $n(i)$, the receipt $r(i)$ and $y(i)$ for each $i \in E$, $[y - x']_t(j_t)$ for each $j_t \in \pi_{1t}(E)$, and $v_t(k_t)$ for each $k_t \in \pi_{2t}(\pi_{1t}(E))$.

Step 3.2 From the column labelled v the EA calculates the tally (which can be verified by anybody). The tally should be equal for all the T copies.

Phase 4: The post-election audit

The EA could try to cheat by changing the values in column v, so an audit is necessary. This is the reason we made T copies of the audit table. This is one of the solutions presented by Punchscan ([6], section 5.4), which we adopt here too. Altrenatives are discussed in Section 7.

Step 5.1 The Auditors create T random bits $b_1 \dots b_T$: Left or Right.

Step 5.2 For $t \in \{1, \dots, T\}$:

- **Left** If $b_t =$ Left, then for each j the EA opens $i := \widehat{\pi_{1t}^{-1}(j)}$ and $\widehat{x'_t(j)}$, and it is verified whether $y_i - x'_t(j) = [y - x']_t(j)$ holds.
- **Right** If $b_t =$ Right, then for each j the EA opens $k := \widehat{x''_t(j)}$ and it is verified whether $[y - x']_t(j) = x''_t(j) + v_t(k)$ holds. Observe that this equation should be satisfied because for the same ballot $y = x + v = x' + x'' + v$ so $y - x' = x'' + v$.

5 Formalizing the properties of our protocol

5.1 Assumptions

We now state the assumptions under whcich we will prove the security of our protocol.

ASSUMPTION A *We assume that the Election Authority uses a bit commitment scheme that is unconditionally hiding.*

ASSUMPTION B *We assume that the Election Authority cannot break the computational assumption of the computationally binding bit commitment scheme before the election ends.*

ASSUMPTION C *We assume that the ballot receipts given to the voter are unforgeable and undeniable.*

ASSUMPTION D *We assume that, except from the ballot image and the information committed to in the audit table, no other copies of (the information on) the ballots exist.*

ASSUMPTION E *We assume that the challenge bits of the Auditors are truly randomly.*

ASSUMPTION F *Though the Election Authority may try to cheat, we assume that it will collaborate in all the steps of the protocol.*

Assumptions A and B are extensively discussed in Section 6. Assumption C can be satisfied using special paper (with watermarks etc) to print the ballot receipts. In parctice D may be the hardest assumption to fullfill if the number of ballots is large. It can be satisfied by putting the ballots in transparent envelopes, for instance. Assumption E can be provably assured by the proper use of hash functions, using them on data that even the Auditors do not control.

5.2 Theorems

THEOREM 1 (PRIVACY OF THE BALLOT)
If Assumptions A through F are satisfied, then the protocol presented in Section 5 has the following general characteristics:
a) When the voter leaves the booth, the receipt that the voter retains leaks no information about her vote.
b) The additional data revealed during the post-election audit reveals no information about individual votes.
Combined with (a) this implies that our protocol guarantees the privacy of the voter.

PROOF: (a) When the voter leaves the booth, her receipt contains a mark in row $y(i)$ and the serial number of the ballot, i. Since $y(i) = x(i) + v(k) (\mathrm{mod}\ m)$ and $x(i)$ is chosen at random, no information is revealed as long as no information about $x(i)$ is revealed. But this is true since the value of $x(i)$ in the table is committed to by using an unconditional bit commitment scheme.
(b) During the post-election audit, for each t either x'_t and π_{1t}^{-1}, or x''_t and π_{2t} are opened,. Since these values are chosen randomly to satisfy the equation $x(i) = x'_t(j) + x''_t(j) (\mathrm{mod}\ m)$, knowing only one of them does not give information about $x(i)$. In addition, during the post-election audit either π_{1t}^{-1} is opened or π_{2t}, but never both, so that the overall permutation π_t between the left and the right part of the audit table is never revealed. ◄

THEOREM 2 (INTEGRITY VOTE COUNT)
a) For each ballot that was constructed and printed dishonestly during Phase 1, the Election Authority has a probabilty of at least 1/2 of being caught.
b) If the Election Authority did not cheat during Phase 1 but during Phase 3, it will be caught with probability at least $1 - (1/2)^T$.

PROOF: (a) This follows from the fact that a total of $2S$ ballots are being printed, and that half of them are audited as described in Steps 1.4, 1.5 and 1.6.
(b) In order to rig the vote count, the EA must cheat on x'_t and π_{1t}^{-1}, or on x''_t and π_{2t}, for each of the T copies of the audit table. So the EA only gets away with cheating if it successfully predicts the coin flips in Step 4.1 which, under assumption E, only happens with $p = (1/2)^T$ ◄

5.3 Other requirements for security

Though the theorems of the previous subsection capture the essential properties of our protocol and seem quite convincing, we would like to verify that we have not overlooked any requirement for a secure voting protocol. We therefore check the list of requirements from [8], to which we refer for discussion and justification.

REQUIREMENT A (ONLY VALID VOTERS) *Only persons on the valid voter list, called voters, can create a ballot and deposit it in the ballot box.*

REQUIREMENT B (ONE MAN ONE VOTE) *A voter can cast at most one vote.*

Our protocol does not address these issues; we assume that existing procedures are used.

REQUIREMENT C (PRIVACY INDIVIDUAL BALLOT) *Filling in the ballot and putting it in the ballot box is a confidential act, and under no circumstance, not even with the connivance of the voter, should an outsider be able to deduce for whom or for what the voter casted her vote.*

This requirement follows from Theorem 1.

REQUIREMENT D (INDIVIDUAL VERIFIABILITY) *The voter (i) can verify that she created a valid vote, (ii) can revise her vote before casting it, and (iii) can convince herself that her vote is faithfully included in the set of votes tallied.*

Subrequirement D(ii) is fulfilled by determining that a voter unsatisfied with her vote can throw away her ballot and ask for a new one.

Subrequirement D(i) is true provided that the voter can trust that the offset printed on the ballot, $x(i)$, corresponds with the value in the audit table. But this follows from Theorem 2 part (a).

Assumption C implies that if a voter does not find her receipt published in te audit table, she has a proof that the EA is trying to cheat. So assuming that the EA publishes all receipts, the voter can check her value y on web site of the EA showing the audit table. The fact that with high probability her encoded vote y on the left is transformed into the correct value v on the right follows from Theorem 2 part (b). This shows that subrequirement D(iii) is fullfilled as well.

REQUIREMENT E (BALLOT BOX SECURITY) *During the voting session it should be possible neither (i) to see the vote on any ballot deposited in the ballot box, (ii) modify a ballot, (iii) remove a ballot from the ballot box, nor (iv) add ballots not coming from voters.*

Subrequirement (i) follows from part (a) of Theorem 1. Subrequirements (ii) and (iii) are satisfied since each voter gets a receipt and can complain if a ballot has been modified or removed. Subrequirement (iv) is satisfied since each vote in the table has a voter id $n(i)$ associated to it.

REQUIREMENT F (COUNT INTEGRITY) *All valid votes encountered in the ballot box, and only those, will be included in the count.*

This follows trivially from the proofs of the previous requirement, E.

REQUIREMENT G (PUBLIC VERIFIABILITY) *The tallying of the votes happens in a manner such that anyone can verify that Requirements C, E and F hold.*

This is ensured by the use of the ballot receipts which need not be kept secret, and by the use of the audit table, which is public.

Requirement H (robustness) *The process has a very high probability to succeed.*

The only way the process cannot complete is if the EA refuse to open the necessary bit commitments, but this contradicts Assumption F.

6 On unconditional privacy and computational integrity

6.1 Unconditional or everlasting privacy

Most election systems so far, with the notable exception of [14] and [15], have the property that the privacy of the ballot is only computational. This flaw is really worrisome for the following reason: with storage becoming cheaper and cheaper every year, we *must* assume that all data that has been made public through an election protocol, will never be erased, i.e. that some copy of it will be stored forever. We also must assume that at some point in the future it will be possible to break the underlying computational assumption, and then it will become public who voted for whom.

Though one can argue that this information might have become irrelevant after many decades, this point is more important than it seems. For instance, people might like to know who the President of the United States voted for when he was young. He might have had a flirt with the communist party, who knows. Even today historians will find it interesting to know Churchill's voting behavior in 1900, when he was about 25 years old. A more dramatic example (due to an anonymous referee) is a scenario in which a dictator gets elected democratically after decades of trying. Once in power, he systematically goes after the voters who voted against him in earlier elections, or after their descendents.

Real world voting systems always have had the property that the vote (the information containing the voter's choice) is irretrievably destroyed. Newly proposed protocols should have this property too. **Computational privacy is not enough for voting**, since one day or another the computational assumption will be broken.

The only cryptographic primitive used in the protocol proposed here and in PunchScan is a bit commitment scheme. However, instead of using encryptions as bit commitments, as proposed in PS[5], section 9, it is more interesting to consider using bit commitments that are unconditionally hiding and computationally binding.

The result is an election scheme that has computational integrity, while the privacy is unconditional or everlasting. Though some people argue that the opposite (unconditional integrity and computational privacy, is preferable, we do not think so because in order to change the outcome of the election, the authorities would have to break the computational assumption **while the election is going on**. Though it is very hard to estimate how difficult it is to break a system 50 years from know (computational privacy), it very well possible to create a system based on a computational assumption that won't be broken in the next three months (computational integrity).

6.2 How to prevent cheating on the computational binding bit commitments

Note that a computationally binding bit commitment can be split over various authorities of which a certain threshold must cooperate in order to open the BC. Such schemes are known as verifiable secret sharing. From a cryptographic perspective this is probably the preferred solution.

However, in some instances it is not clear that the authorities can share the power to open the BCs with others. In Brazil, for instance, one specific organization is by law responsible for conducting the election, and it cannot share secrets with other entities because that could make it a hostage of them ("Do what we say otherwise we won't open our secrets and the elections will not terminate"). This is one motive for the solution outlined below.

A second motive for the scheme proposed below is the following line of thinking. It is known that unconditionally hiding and unconditionally binding BCs are possible under additional assumptions, for instance physical assumptions about the channel between the parties, or assumptions that one party has limited computational resources, like a limited processor, or a limited memory size. Here we propose a BC scheme in which we make the following assumptions:

1. The computer hardware used by the committing party has limited processor capacity, and it is possible to define computational or cryptographic one-way functions for which the probability of the processor inverting it during a d day period is truly negligible. Here d should be bigger than the time between the pre-election and post-election cerimony, which is typically a couple of months.

2. This hardware is held in a safe place between the pre- and the post-election audit in a way that *(a)* nobody can touch it, and (*b)* witnesses are allowed a permanent view of the equipment to make sure that nobody touches it.

Under these assumptions we obtain a voting protocol with unconditional (everlasting) privacy and in which cheating on the (computational) integrity of the vote count is impossible.

In general, cryptographers should be warry of assumptions about secure hardware, but on the other hand we must recognize that they can be useful. For instance, the security of banks and certification authorities heavily rely on them. In other words, this is known technology that *might* have its place in voting as well.

6.3 A more detailed description

We can even narrow down this idea even more. Suppose that, to create the random strings for opening the bit commitments, one uses a Hardware Security Module (HSM) which is connected to a computer. An HSM is funcionally equivalent to a smart card but bigger, with more security features and higher processing capacity

A few weeks before the elections, a cerimony is held to create the BCs. After this pre-election cerimony, computer and HSM are kept in a "transparent" room so that any observer can see it sitting there from an adjacent "viewing" room, and verify that nobody touches it. The smart cards used by authorities to activate the HSM are also held in transparent boxes, next to the HSM. All equipment is being guarded by heavily armed people, and political parties are allowed permanent access to the "viewing" room so they can organize a 24 hour vigilance, checking that nobody is touching. After the election is over, a post-election cerimony is held in which the BCs requested by the auditors are opened, for all others BCs the random strings to opening them are destroyed.

In this approach the computer inside the transparent room cannot rig the election because it has insufficient computational power to break the computational assumption that guarantees the binding property of the BCs (apart from all other safety features present). And even the NSA cannot rig the election since it cannot access the data inside the HSM/trusted computer.

Note that the observers do *not* have to verify the correctness of the software that is being used. As long as they verify *(1)* that nobody can access the machines between the two cerimonies, and *(2)* that the bit commitments that are opened are correct, they can trust the outcome of the election. (But the authorities, of course, have an interest in the correctness of the software to make sure that the election completes successfully.)

7 More on the audit procedures

The audit procedures described in Section 4 are the simplest to make the protocol and proofs work, but some additional comments are appropriate.

In the first place, unused ballots could be opened and audited after the election. Similarly, it is possible to give each voter two ballots and have the voter choose: one ballot will be used for auditing, the other for voting.

In addition, other auditing protocols could be used. Apart from the approach used here, PunchScan presents another (standard) approach ([5] section 7.6, [6] section 5). It is the equivalent of one round of randomized partial checking as presented in [16]. Unlike the mechanism presented before there is only one copy of the table, and for each j in the middle part of the table a random bit is created out of EA's control: Left or Right, which has the following semantics:

Left: The EA opens $i := \widehat{\pi_1^{-1}(j)}$ and $\widehat{x'(j)}$, and it is verified whether $y(i) - x'(j) = [y - x'](j)$ holds.

Right: The EA opens $k := \widehat{x''(j)}$ and it is verified whether $[y - x'](j) = x''(j) + v(k)$ holds.

Using this approach we catch a cheating EA with probability $1/2$ for each vote $v(k)$ he modifies. However, information is revealed about the overall permutation $\pi = \pi_2 \circ \pi_1$ between the left and the right part of the table. In particular, the size of privacy set is reduced by about a factor of two (exactly two if we require that the number of Left and Right bits are equal), which can be problematic when there is a small number of voters.

One way out is the the following: instead of using two permutations π_1 and π_2, we use four. We also split x in four parts: $x = x_1 + x_2 + x_3 + x_4 (\mathrm{mod}\ m)$ and use two rounds of randomized partial checking. The audit table now has five parts and essentially looks like this:

$\widehat{x}$ y	$\widehat{x_1}$ $y - x_1$ $\widehat{x_2}$	$y - x_1 - x_2 = x_3 + x_4 + v$	$\widehat{x_3}$ $x_4 + v$ $\widehat{x_4}$	v

It is obvious how this solution generalizes to an arbitrary number of permutations.

Alternatively we could use Chaum's improvement [2] of the mixing protocol proposed in [16], which used four permutations. See [3] for a detailed description. However, for both methods, the original randomized partial checking and Chaum's improvement, a rigorous quantification of the amount of information revealed about the overall permutation π is still lacking. In [17] it is proven that a constant number C of auxiliary permutations is sufficient to get it below any ϵ, but how small C can be in practice is still unknown. This is subject of further research.

8 Conclusion

This paper started as a comparison of Prêt-à-Voter and Punchscan. Surprisingly, we found a third variations which seems an improvement on both: the simpler ballot layout of PaV with the simpler cryptography of PS. Cryptographically speaking, PunchScan is actually slightly more complicated than the variant presented here. The fact that in PS *both* the top and the bottom layer can be flipped seems to complicate matters without a clear benefit.Also in terms of ballot lay-out there is virtually no difference between PaV, PS, and the variant presented here.

As our analysis shows, the three protocols are so similar that they can be considered variations on the same theme. Their strong points are their simplicity, together with the fact that they can provide inconditional privacy. Their most serious drawback is that they are subject to the random-spoil attack: an outsider forces the voter to bring back a receipt with a vote in a specific location y. Though the outsider cannot impose a vote for a certain candidate, he is preventing that the voter

casts a vote in favor of some candidate; her vote is effectively randomized. At the moment we see no solution other than offering the voter the choice between several ballots.

Acknowledgments

David Chaum and I discussed PunchScan several times, though we never went over the crypto aspects. Ben Hosp and Stefan Popovenuic very kindly provided me with [6], and the first clarified some other doubts that remained. With respect to the ballot lay-out, Peter Ryan observed that complex parallel constructions can be avoided simply by putting one sheet on top of the other.

References

[1] J. Benaloh and D. Tuinstra, "Receipt-free secret-ballot elections," in *Proceedings of the 26th ACM Symposium on Theory of Computing*, pp. 544–553, 1994.

[2] D. Chaum, "Secret-ballot receipts: True voter-verifiable elections," *IEEE Security and Privacy*, vol. 2, pp. 38–47, January/February 2004.

[3] J. Bryans and P. Ryan, "A dependability analysis of the Chaum Voting Scheme," Tech. Rep. CS-TR: 809, School of Computing Science, University of Newcastle, October 2003.

[4] D. Chaum, P. Ryan, and S. Schneider, "A practical, voter-verifiable election scheme," Tech. Rep. 880, Newcastle University, School of Computing Science, Dec 2004.

[5] S. Popovenuic and B. Hosp, "An Introduction to Punchscan." Version dated Oct 15, 2006. http://punchscan.org/papers/popoveniuc_hosp_punchscan_introduction.pdf, 2006.

[6] B. Hosp and S. Popovenuic, "Punchscan Voting Summary." Version dated Feb 13, 2006, obtained from first author, 2006.

[7] B. Adida, *Advances in Cryptographic Voting Systems*. PhD thesis, Massachusetts Institute of Technology, 2006.

[8] D. Chaum, J. van de Graaf, P. Y. A. Ryan, and P. L. Vora, "High integrity election." http://eprint.iacr.org/2007/270, 2007.

[9] P. Ryan, "A variant of the Chaum voter-verifiable scheme," Tech. Rep. CS-TR: 864, School of Computing Science, University of Newcastle, October 2004.

[10] J. van de Graaf, "Merging Prêt-à-Voter and PunchScan," in *Anais do VII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pp. 207–210, Sociedade Brasileira de Computação, 2007.

[11] A. Neff, "Practical high certainty intent verification for encrypted votes." http://www.votehere.net/vhti/documentation/vsv-2.0.3638.pdf, 2004.

[12] R. Rivest, "The ThreeBallot Voting System." Version dated Oct 15, 2006. http://people.csail.mit.edu/rivest/Rivest-TheThreeBallotVotingSystem.pdf, 2006.

[13] R. Araújo, R. Custódio, and J. van de Graaf, "A verifiable voting protocol based on Farnel." IAVoSS Workshop On Trustworthy Elections (WOTE2007), Ottawa, Canada, 2007.

[14] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung, "Multi-authority secret-ballot elections with linear work," *Lecture Notes in Computer Science*, vol. 1070, pp. 72–81, 1996.

[15] T. Moran and M. Naor, "Receipt-Free Universally-Verifiable Voting with Everlasting Privacy." CRYPTO 2006, 2006.

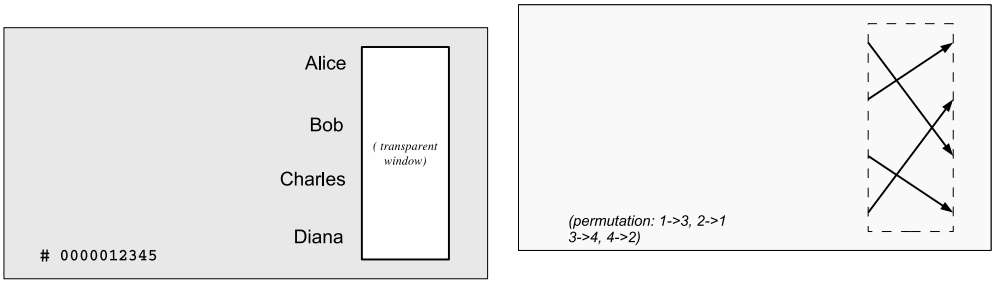
[16] M. Jakobsson, A. Juels, and R. L. Rivest, "Making mix nets robust for electronic voting by randomized partial checking," in *Proceedings of the 11th USENIX Security Symposium*, pp. 339 – 353, 2002.

[17] M. Gomulkiewicz, M. Klonowski, and M. Kutylowski, "Rapid mixing and security of Chaum's visual electronic voting," in *ESORICS '03*, LNCS 2808, pp. 132–145, Springer-Verlag, 2003.

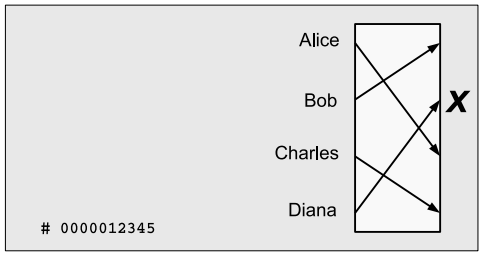
A Comparing the ballot layouts of PaV and PS

A.1 The differences between the PaV and PS ballot layout

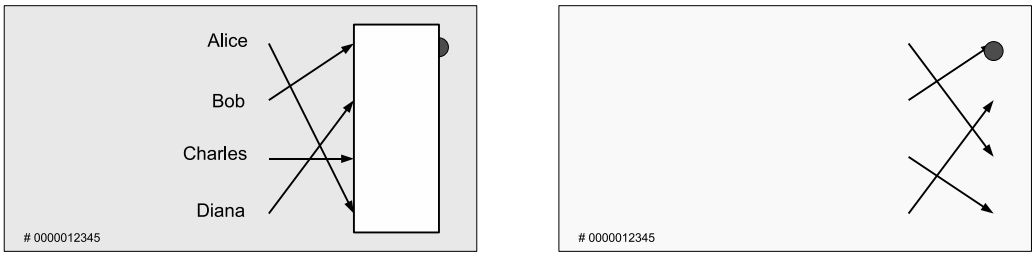
The equivalent of the Prêt-à-Voter ballot presented above then has a top layer (the former left side) with a transaparent window, and a bottom layer (the former right side):



A vote for Diane is a mark on the top layer; the bottom layer will be destroyed:



Likewise we can use arrows for the PunchScan ballot; now we need arrows on both layers. We show a ballot which already contains a vote for Diane and of which the two layers have been separated already. Note that both layers contain a part of the mark, that the user can choose which layer he wishes to keep and that neither layer reveals information about the vote.



In conclusion, this section shows explicitly that the auxiliary symbols in PS (X, Y, Z, T in our example) allow the use of two permutations, whereas PaV uses only one permutation.

A.2 The similarities between the PaV and PS ballot layout

The formar subsection had as purpose to stress the differences between PaV and PS, which has some cryptographic implications. Here we want to stress the similarities with respect to the ballot layout. In particular, it might seem that cutting the vote in two as in PaV severely limits the ballot lay-out of PaV. We will show some simple trick to get around this apparent limitation. Our solution will

show how to have multiple races on the ballot, or how to enter a several digit number. (This is the way voting is done in Brazil: each candidate gets a number assigned for the election. Numbers are between 2 and 5 digits long, depending on the race.)

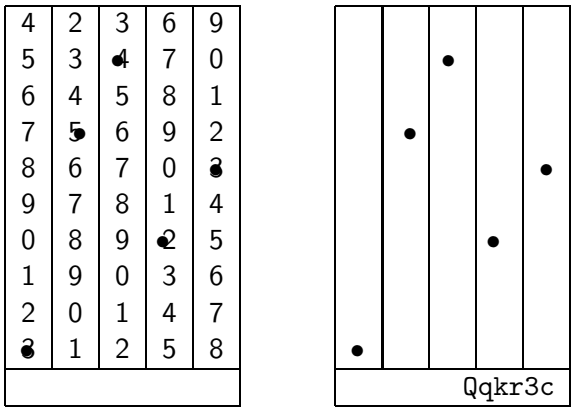
For the initial idea please look at the following ballot lay-out; once you get the idea the generalizations are straightforward.

Race 1	**Race 2**	**Race 1**	**Race 2**
3: Buddhist	1: Einstein	X	
0: Anarchist	2: Newton		
1: Alchemist	0: Bohr		X
2: Nihilist			
(Offset $x = 1$)	*(Offset $x = 2$)*	`Qqkr3c`	`bkuryt`

This ballot shows two different, unrelated races: one in the first and third column, and one in the second and fourth. Note that there are two encryptions in the bottom row of the third and fourth column. This allows that the right part of the ballot be split again between the third and fourth column, thus hiding any correlations between the two different races. If this is not necessary, then one encryption containing the two offset values would be sufficient.

So we see that the hardest part is not the ballot lay-out, but finding a way to help the voter. There are many options; we will outline one for concreteness. Suppose that all the four columns have the same width. Imagine now a mask that vertically shows only the first column. In addition there is a horizontal ruler which highlights one cell in the first column and has a whole at the same height in then third column, through which the voter can write an X or perforate a hole. Then by shifting the mask one column to the right (or the ballot to the left), the second and fourth column are serviced.

Alternatively, we can put the left part on top of the right part. So instead of a left and a right part, we suppose two equal-sized pieces of paper. What was the left now goes on top, while the right part constitutes the bottom, and we suppose that the voter casts his vote by perforating a hole through both layers simultaneously. As before, the top layer contains a permuted version of the candidate list and will be destroyed afterwards, while the bottom layer contains an encryption of this permutation and will be retained. The idea is outlined in the following diagram, which showes the upper and lower layer of a vote for candidate 35423:



Indeed, this is very similar to PunchScan, except that there is no permutation on the bottom part. In other words, here the bottom sheets are identical for everybody.

A serious drawback of perforating a hole in both layers is that the top (former left) part now contains the full information of the vote, which is undesirable. One solution is to reverse the order of the sheets: the right part, made of some transparent material, goes on top, on which the voter marks his votes. By using glass between the sheets it can be avoided that any trace of the voter's choice

remains on the bottom (left) sheet. In fact, glass with pre-drilled dips could be used for perforation, avoiding that the bottom layer is affected.

To reduce voters mistakes, we could also use a DRE-like machine. The left sheet, now made of transparent material, is put on top of the screen of the DRE. The latter understands about the general ballot lay-out (the exact positions of the 5 columns with 10 rows) and the restrictions (exactly one mark per column) and so can guide voter when she marks here choices. However, the DRE does **not** know what the voter voted for since this is encoded in the top sheet. Once the voter is done, the screen image is printed and the voter puts the transparent sheet over the printout for a final verification. If confirmed, the top sheet is destroyed while the bottom sheet is kept by the voter as a receipt. The DRE keeps the information about the cells chosen which it forwards to the tallying authority. A little detail here is the ballot ID, which the DRE must learn. One possibility is to have a barcode sticker on the (top) sheet, which will be read by the DRE before the voter begins and which is printed on the receipt. As one of the last steps, the sticker is peeled off the top sheet and stuck just above its image printed on the printout of the bottom sheet, allowing for quick visual verification to make sure that the DRE copied the barcode faithfully.

B A brief description of Punchscan

The header of the table used by Punchscan is as follows:

i	$\widehat{x_1}$	$\widehat{x_2}$	y	$\widehat{j}$	$\widehat{t_1}$	$y - t_1$	$\widehat{t_2}$	$\widehat{\pi_2(j)}$	v
	$P_{.1}$	$P_{.2}$	$P_{.3}$	$D_{.1}$	$D_{.2}$	$D_{.3}$	$D_{.4}$	$D_{.5}$	$R_{.1}$
$\ldots$									

The first row shows the notation introduced in this paper, whereas the second row shows the notation of [6] and [5]. Observe that where they use x, y, z as the indices of the left (P), middle (D) and right (R) part of the table, we use i, j, k, so that when they we write (x, P_3) we would write y_i, etc. Also, the description of Punchscan uses $m = 2$, so that adding $1 (\mathrm{mod}\ 2)$ is called "flipping" or "inverting" the bit.

Simplifying this table by defining $x_1 = t_1$; $x_2 = t_2$ is tempting but leads to an *insecure* protocol because of the following difference between PaV and PS. In PaV the offset (or offsets, in the new protocol) is (are) kept secret: the left side of the ballot is destroyed, and the value on the right side is protected by a bit commitment. But in Punchscan the offset from the top (x_1) or bottom (x_2) layer can be deduced from the printed ballot. One layer gets destroyed but the other has its scanned image published, so this information, combined with the information about the destroyed layer revealed during the post-election audit, compromises the ballot security, which happens with $p = 1/2$. Therefore x_1, x_2, t_1 and t_2 are chosen randomly satisfying $x_1 + x_2 = t_1 + t_2 (\mathrm{mod}\ m)$.