Merging Prêt-à-Voter and PunchScan

Jeroen van de Graaf

[1]Laboratório de Computação Científica – Universidade Federal de Minas Gerais (UFMG)
Av. Antônio Carlos 6627 – 31270-901 – Belo Horizonte (MG) – Brasil

jvdg@lcc.ufmg.br

Abstract. *We describe a variation of Prêt-à-Voter that keeps the same ballot layout but borrows and slightly modifies the underlying cryptographic primitives from Punchscan, substituting the mix network for bit commitments.We also suggest a limited-resources solution to get to unconditional privacy and unconditional integrity, and propose ways to have several races on the Prêt-à-Voter ballot.*

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 him to verify that her vote is included in the tally without revealing any usefull information about her vote. Though this idea is not new, Chaum's paper [2] arguably gave a new impetus to this line of research (see also [1]).

Chaum's paper was improved upon in two significant ways. First there is a protocol called Prêt-à-Voter (PaV), as described in [3], which has several advantages over [2], such as a simpler ballot lay-out, 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, etc. However, PaV still uses decryption mixing. Inspired by this, Chaum developed PunchScan (PS). See the site *www.punchscan.org* for fancy demos. For a detailed protocol description we refer to [7] and [5]. PS differs from PaV in several aspects: *(1)* in each ballot both the top and bottom layer are permuted; *(2)* a mark is placed on both layers; *(3)* the voter gets to choose which layer he keeps and which gets destroyed; *(4)* no mixing takes place; the only cryptographic primitive needed is a Bit Commitment scheme.

In this paper we obtain a new protocol by merging PaV and PS as follows: we maintain PaV's ballot lay-out 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.

The outline of this paper is as follows: we start with a high-level description of the PaV ballot, but instead of using mixing we describe how to apply the underlying cryptographic ideas used in PS to PaV. We also propose some improvements to the protocol, and provide a very brief description of the cryptography of Punchscan. We assume that the reader is familiar with the general setting and the terminology of voting protocols.

2. The Prêt-à-Voter Ballot

The ballots used in PaV are described in detail in [3], section 4, using an example with 4 candidates. A base canonical ordering of candidates is defined: 0: Anarchist, 1:

Alchemist, 2: Nihilist, 3: Buddhist. An example ballot looks like this (section 4.1):

3: Buddhist	X
0: Anarchist	
1: Alchemist	
2: Nihilist	
(Offset $x = 1$)	`Qqkr3c`

The left part contains a cyclic permutation (shift) of the candidates[1]; in this case the offset $x = 1$.The right part is empty except for the last row, and the voter votes by putting an "X" 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 X and the encryption of the offset are associated to the ballot image. The voter can take the right column home as a receipt. At the end of the day, all ballots will enter the mix process. That is, each mix contributes in decrypting the shift and shuffling all the ballots; see section 6 of [3].

3. Using bit commitments instead of mixing

Mixing is a tedious process and has several disadvantages: it is difficult to explain to the average person, the privacy of the ballot is only computational, it is computationally intensive, etc. A protocol that uses bit commitment does not have these disadvantages: pieces of papers in an enveloppe serve as an excellent explanation for BCs, unconditionally hiding bit commitment schemes exist, and they are certainly not less efficient than mixing. Therefore our purpose here is to develop a variant of PaV using BCs.

Since Punchscan uses two permutatons, both the top and the bottom layer, a straightforward idea is to break the offset value x in two, i.e. to choose x_1 and x_2 random such that $x = x_1 + x_2 (\mathrm{mod}\, m)$. We let the Election Authority(EA) commit to x_1 and x_2. We write these BCs at the bottom of the right column on the ballot (like in PaV) or, alternatively, we have the EA commit to these values publicly and use a unique ballot id number to establish the link between the two BCs published and the printed ballot.

Furthermore we use the following notation: x is the offset; y is the number of the row marked by the voter, counting from 0 to $m - 1$; v is the actual vote, that is, the row chosen in 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 table to be created by the Election Authority (EA) before the election which is a simplification of 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, $y - x_1$ and v will remain empty until the counting of the votes, as we will see below.

[1]We restrict ourselves to cyclic permutation for simplicity of exposition; unless noted otherwise our protocols extend to full permutations.

i	y	$\widehat{j}$	$\widehat{x_1}$	$y - x_1$	$\widehat{x_2}$	$\widehat{\pi_2(j)}$	k	v
1								
$\cdots$								
$\cdots$								
$2R$								

Observe that the table is divided in a left, middle and right part, with i, j and k as index, resp. Let π_1 be the permutation between the rows of the first and the second part, and π_2 between the rows of the second and the third part. Then the columns labelled $\widehat{j}$ and $\widehat{\pi_2(j)}$ are used to define and to verify these two permutations. In particular, j should be the pre-image π_1^{-1} under π_1 for row j, whereas k contains the image $\pi_2(j)$ under π_2 for row j.

Auditing the ballot construction Let there be $2R$ rows. The set of rows is divided randomly in an audit set A and election set E both of size R. The EA is now required to open 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 in the middle part of the table if $j = \pi_1(i)$ and $i \in A$. The right part contains no commitments. Scrutineers should check that all bit commitments were created honestly. After the audit, the EA prints the unopened ballots with index $i \in E$.

The election The voter casts her vote as described earlier, and for each vote the value y_i is determined. Since EA also knows x_1 and x_2 he can compute the corresponding values $y_j - x_{j1}$ and v_k.

Publishing the results After the election, the EA publishes y_i for each $i \in E$, $y_j - x_{j1}$ for each $j \in \pi_1(E)$ and v_k for each $k \in \pi_2(\pi_1(E))$. From the column labelled v he calculates the tally, which can be verified by anybody.

Auditing the votes published The EA could try to cheat by modifying the values v_k. We therefore first define the following *naive* approach: 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 sematics:

Left The EA opens $\widehat{x_{j1}}$ and it is verified whether $y_{\pi_1^{-1}(j)} - x_{j1} = (y - x_1)_j$ holds.

Right The EA opens $\widehat{x_{j2}}$ and it is verified whether $(y - x_1)_j = x_{j2} + v_{\pi_2(j)}$ holds. Observe that this equation should be satisfied because $y = x + v = x_1 + x_2 + v$ so $y - x_1 = x_2 + v$.

Using this approach we catch a cheating EA with probability $1/2$ for each vote v_k he modifies. However, too much information is revealed about the overall permutation $\pi = \pi_2 \oplus \pi_1$ between the left and the right part of the table, violating voter privacy. We can think of three possible ways out:

(1) We do K versions of this protocol in parallel, each with different bit commitments and one Left/Right choices for all rows in each parallel version. Then the probability of EA getting away is 2^{-K}. This is the solution adopted by Punchscan ([5], section 5.4).

(2) Instead of using two permutations π_1 and π_2, we use four. We also split x in four parts: $x = x_1 + x_2 + x_3 + x_4 (\mathrm{mod}\ m)$. Then we use Chaum´s improvement [2] of the mixing protocol proposed in [6]. See [1] for a detailed description.

(3) We use a special kind of bit commitment scheme that has a homomorphic property: we assume that the multiplication of two bit commitments is equivalent to the addition $(\mathrm{mod}\ m)$ of their contents. BCs with this property can be constructed from from homomorphic encryption schemes. This variant does not trivially generalize to elections in which a mere cyclic shift will not do and full permutations are needed.

4. A brief description of Punchscan

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

i	$\widehat{x_1}$	$\widehat{x_2}$	y	$\widehat{j}$	$\widehat{t_1}$	$y - t_1$	$\widehat{t_2}$	$\widehat{\pi_2(j)}$	k	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 [5] and [7]. 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 $x_1 + x_2 = t_1 + t_2(\mathrm{mod}\ m)$.

5. About unconditional integrity and unconditional privacy

The only cryptographic primitive used in PunchScan, and now in Prêt-à-Voter too, is a bit commitment scheme. However, instead of using encryptions as bit commitments (as proposed in PS), it may be 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).

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 entity is by law responsible for organizing 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 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 p day period is truly negligible. Here p 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 during these p days in a way that *(a)* nobody can touch it, and (*b)* witnesses are allowed permanent access to make sure that nobody touches it.

A more detailed description

We can even narrow down this idea even more. Suppose that, to create the keys that open the bit commitments, one uses a Hardware Security Module (HSM; basically the equivalent of a smart card but with more security features and higher processing capacity) which is connected to a trusted computer.

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 froma 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 the required BCs are opened, for all others the keys to opening them are destroyed.

In this scheme, 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.)

Note that most cryptographers (and I consider myself one of them) usually don't like special equipments (and it is our job not to like them), but we should recognize

that, for instance, all of PKI is based on them: VeriSign stores its root certificates in an HSM held in a well-protected strong-room in a well-protected building. This is known technology that could have its place in voting.

Moreover, from a psychological point of view, with HSM-like solutions it may be easier to win the trust of the lay-man, since they are easier to grasp than advanced cryptographic protocols. When designing systems that need to be trusted by a large majority of the population, such as voting, these kind of arguments play an important role. A sophisticated voting protocol that I cannot explain to my colleagues in the computer science department over a cup of coffee is probably doomed. A simple, understandable, robust method of making a system secure may be preferable.

6. More flexible ballot lay-outs for Prêt-à-Voter

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.

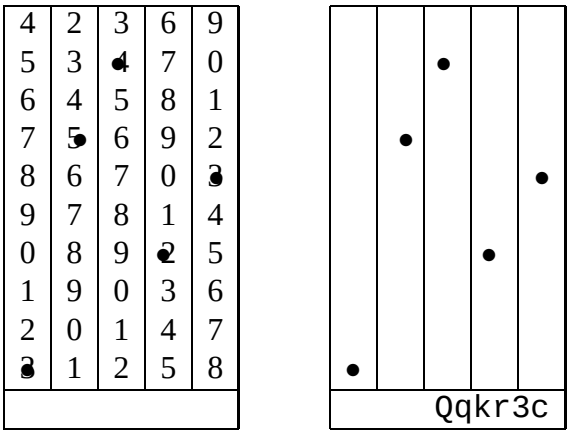
3: Buddhist	1: BBB	X	
0: Anarchist	2: CCC		
1: Alchemist	0: AAA		X
2: Nihilist			
(Offset $x = 1$)	*(Offset $x = 2$)*	`Qqkr3c`	`bkuryt`

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 column, we suppose two equal-sized pieces of paper. What was the left column now goes on top, while the right column 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:



This is very similar to PunchScan, except that there is no permutation on the bottom layers. I.e. 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: putting the right part, made of some transparent material, 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 "wells" (i.e. not all the way through) could be used for perforation avoiding that the botom layers is affected.

To avoid 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 knows about the general lay-out of the ballot (the exact positions of the 5 columns with 10 rows) and the restrictions (exactly one mark per column) and so can guide voter. 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 exact screen image is printed and the voter puts the transparent top sheet on top of 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. A little issue 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 receipt. As one of the last steps, the sticker is peeled of the top sheet and sticked just above its image printed on the printout of the bottom sheet, allowing for quick visual verification.

7. Conclusion

This paper started as a study the similarities and differences between Pret-a-Voter and Punchscan. Surprisingly we found a merge which seems an improvement on both. The Section 4 shows that cryptographically PS is actually slightly more complicated than the PaV variant presented in Section 3. The fact that *both* the top and the bottom layer can be flipped seems to complicate matters, without any real benefit. In particular, the choice between top and bottom layers does not seem to add to the security of Punchscan, unlike in [2]. Also in terms of ballot lay-out there seems little difference between the two schemes.

History and shortcoming of this version

This paper is the result of a long-overdue task I had set myself: going to the cryptographic core of both Prêt-à-Voter and PunchScan. This effort resulted in Sections 1 thru 4 which, together with the references, were submitted as a short paper to the Simpósio Brasileiro de Segurança. Drafts of Sections 5 and 6 were written at the same time, but they did not fit; they are included now for completeness.

Intended as a short paper (4 pages only), I am aware of the short-comings of the current version: if you do not know the subject already (or without reading the cited papers) it is almost incomprehensible, and there are not many references. Furthermore, a more detailed comparison of the differences (e.g. encryption printed on the ballot vs. ballot ids printed + bit commitment elsewhere), including a table listing them, is still missing.

Acknowledgments

Ben Hosp and Stefan Popovenuic very kindly provided me with [5], and the first clarified some other doubts that remained. Peter Ryan avoided that I make a fool of myself in Section 6 when I proposed to drill two equidistant holes simultaneously in the left and right part, instead of superimposing them. With David Chaum I had various discussions about PunchScan (ballot lay-out with arrows; internet solutions), though we never went over the crypto aspects.

References

[1] Ryan, P.Y.A. *Variant of the Chaum Voting Scheme.* Technical Report CS-TR-864, University of Newcastle, 2004. Also Proceedings of the Workshop on Issues in the Theory of Security(ACM), 2005. pg 81-88.

[2] Chaum, D. *Secret-Ballot receipts: True Voter-Verifiable elections.* IEEE Security and Privacy, 2(1):38-47, Jan/Feb 2004.

[3] Chaum, D., Ryan, P. Schneider, S. *A Practical Voter-Verifiable Election Scheme.* ???.

[4] Crépeau, C., van de Graaf, J. and Tapp, A. *Committed Oblivious Transfer and Private Multy-Party Computation.* CRYPTO'95, Springer, LNCS, vol. 963, 1995, pp.110-123.

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

[6] Jakobsson, M., Juels, A. and Rivest, R. *Making Mix Nets Robust For Electronic Voting By Randomized Partial Checking.* Usenix 2002.

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