Voting with Unconditional Privacy by Merging Prêt-à-Voter and PunchScan

Jeroen van de Graaf

***Abstract*—We present a detailed comparison of the Prêt-à-Voter and Punchscan protocols for booth voting. We also describe a simpler variation that keeps the ballot layout of Prêt-à-Voter but borrows the cryptography from Punchscan, which is based on any commitment scheme. By using unconditionally hiding commitments we obtain a conceptually very simple voting protocol with unconditional privacy.**

***Index Terms*—election protocols, voting protocols, bit commitment, unconditional privacy**

I. Introduction

A. Motivation

Over the last few years we have seen a sequence of papers on voter-verifiable elections. The idea of these systems is to provide the voter with a receipt which, on the one hand, allows her to verify that her vote is included in the tally; but on the other, the receipt does not reveal any information about her choice. Though this idea is not new[1], Chaum's paper [2] arguably gave a new impetus to this line of research (see also [3]). Subsequently, Chaum's protocol was improved upon in two significant ways. First there is a protocol called the Prêt-à-Voter (PaV) protocol, as was described in [4][5]. Shortly afterwards, and inspired by PaV, Chaum developed PunchScan (PS). See the site *www.punchscan.org* for demos and technical descriptions [6][7].

Both protocols have important advantages over [2]: a simpler ballot layout, and pre-printed ballots on which the voter marks his preferences with a pen, thus ensuring that the voting machine 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 learned how a person voted, and protecting thousands of voting machines is very difficult.

There are also some important differences between PaV and PS. First, their ballot layouts are different, and so is the mechanics of voting. But as we will show, these differences are marginal. Second, PaV uses decryption mixing as the underlying cryptographic primitive, whereas PS uses a bit commitment 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 since the commitment primitive is much simpler than the mixing primitive. Additionally, from a theoretical point of view, PS makes a clear link between commitment schemes and voting protocols, with interesting implications for unconditional (or everlasting) privacy.

Jeroen van de Graaf is with the Universidade Federal de Ouro Preto, Brazil.
Manuscript received February 20, 2009; revised August 12, 2009.

B. Unconditional or everlasting privacy

Most election systems so far, with the notable exception of [8],[9] and [10], have the property that they provide only computational privacy of the ballot. This flaw is really worrisome for the following reason: with storage becoming cheaper and cheaper every year, we *must* assume that all data published during an election protocol will never be erased, i.e. that some copy of it survives forever. Sooner or later, the underlying computational assumption will be broken, so eventually 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—perhaps he flirted with the communist party. Even today historians will find it interesting to know Churchill's voting behavior in 1900, when he was about twenty-five years old. More dramatic is a scenario in which a ruthless dictator gets into power after decades of trying. Once in power, he systematically goes after the voters who voted against him in earlier elections, or after their descendants.

Real world voting systems have always had the property that the vote (the information containing the voter's choice) is permanently destroyed. Newly-proposed protocols should have this property too. *Computational privacy is hence not sufficient for a voting protocol*, as the computational assumption is likely to be broken in the future. Though this implies that the correctness of the election outcome is "only" computational, we believe this is a sensible trade-off, since the authorities would have to break the computational assumption *before the election has terminated* in order to alter the election result. Though it is very hard to estimate how difficult it is to break a computational assumption fifty years from now (computational privacy), it is easy to design a protocol based on a computational assumption that will not be broken in the next few months (computational correctness).

C. Summary of results

We present a detailed comparison of Prêt-à-Voter and Punchscan. Then we describe a variation that keeps the simpler ballot layout of Prêt-à-Voter but borrows, and slightly simplifies, the cryptography from Punchscan. Using bit commitments that are unconditionally hiding and computationally binding, we obtain a conceptually simple election scheme that has computational correctness, whereas its privacy is unconditional. An earlier version of this work was published as a four-page short paper [11], which essentially corresponds to the contents of Section IV.

We claim that our protocol has the following properties, all of which, except the first one, are inherited from both PaV and PS:

- **Unconditional privacy of the ballot:** The public view, including all receipts and all other data published on the election web site, reveals no information (in the Shannon sense) about a voter's choice. This implies that, assuming that all unopened commitments are permanently destroyed after the final audit step took place, we obtain everlasting privacy. This property is inherited from PS provided an unconditional commitment scheme is used. PaV uses mixing, thus inherently providing only computational privacy.
- **Coercion-resistance:** As a consequence of the previous property, our protocol is immune to vote buying/selling schemes in the sense that a voter cannot prove whom she has voted for. However, we have no satisfying solution to the randomized coercion attack, in which a coercer forces the voter to bring back a receipt with a mark in a specific location, effectively forcing the voter to cast a random vote.
- **Computational correctness of vote count:** If the election authority passes the post-election audit steps, this means that the vote count is correct, unless an extremely unlikely event has occurred or the authority has succeeded in breaking, before the election ends, the computational assumption on which the commitment scheme is based. This statement is true *independent* of whether the authority tried to cheat and whether the programs that run the election are correct.
- **Individual Voter Verifiability:** Each voter receives a receipt that she can compare with the image published on the election web site. If this image is different or absent, the receipt serves as a proof that the authority is dishonest.
- **Universal Verifiability:** Any observer can verify that the tally has been calculated correctly with overwhelming probability.

D. Comparison to other work

As we advocate the position that privacy should be unconditional, we do not consider voting protocols based on homomorphic encryption and/or (re)encryption mixes which, in fact, constitute the large majority of the voting literature.

To our knowledge, the first voting protocol that provides unconditional privacy was published by Bos in chapter 3 of his thesis[12]. This protocol, which employs the dining cryptographers protocol as an underlying primitive, requires all voters to be online simultaneously and hence is not very practical. For an attempt to remove this limitation, see [13].

In [8], Cramer, Franklin, Schoenmakers and Yung present an elegant protocol with unconditional privacy. Though the protocol presented is tuned towards internet voting (in which each voter is assumed to have his own, trusted computer), this protocol can be recast for booth voting (see [14]).

Moran and Naor have published two papers which also present protocols that provide unconditional privacy. The protocol presented in [9] uses the a voting machine and the ballot casting assurance techniques of Neff and Adida[15][16], resulting in a very different user interface. The protocol presented in the second paper[10] bears some resemblance to the one presented here, which is not surprising since both are based on PunchScan. In fact, it seems that by applying our observations about ballot layout, a simpler user interface for Split-Ballot can be obtained, since both protocols require the voter to perform an addition modulo the number of candidates on the ballot. See [14] for further discussion.

Scantegrity [17] also uses a commitment scheme as the underlying cryptographic primitive, so it can also be modified to provide unconditional privacy. Scratch&Vote [18] strongly relies on homomorphic counters for the ballot layout, and thus can provide computational privacy only.

In [19], Popoveniuc and Vora present a very interesting comparison of various protocols, including PaV, PS and Scantegrity, but they do not address unconditional privacy.

E. Outline of the paper

In Section II we provide more details about commitment schemes, and present the general model of voting. In Section III we provide a detailed description of the differences in the ballot layouts of Prêt-à-Voter and PunchScan, and demonstrate that these differences are insignificant from a practical point of view, despite the fact that the opposite has been argued ([16], page 117).

In Section IV we present our protocol, by merging PaV and PS as follows: we maintain PaV's (simpler) ballot layout but we borrow the underlying cryptography from PS. The final result is superior to both because, compared to PaV, it disposes of mixing, while, compared to PS, it results in a simpler ballot layout.

In Section V we state our assumptions, and present the properties of our protocol. Section VI presents variations and extensions of our protocol, whereas Section VII demonstrates how the overall number of commitments used in the protocol can be kept fairly small.

II. Preliminaries

A. Commitment schemes

A commitment scheme is a cryptographic primitive that implements the equivalent of the following functionality. In the *commit* phase, a Sender writes a certain string, x, on a piece of paper, which he puts in an opaque envelope; he then seals the envelope and puts it on the table. In the *decommit* phase, which is optional, the Sender opens the envelope and shows the text written to the Receiver(s).

Two security properties are required of commitment schemes. The *binding* (or correctness) property asserts that the Sender cannot change his mind by opening a different string $x' \neq x$, thus protecting the Receiver. The *hiding* (or privacy) property asserts that the Receiver cannot derive any information related to the string before the envelope is opened, thus protecting the Sender.

Commitment is a very important and well-studied cryptographic primitive, see for instance [20] and references therein.

It is a well-known fact that if the Sender and the Receiver are connected through an error-free channel, then commitment schemes come in two flavors: computationally hiding and unconditionally binding, or unconditionally hiding and computationally binding. Since our goal is to obtain voting protocols which are unconditionally private, we are only interested in the latter.

In all known practical implementations, the Sender uses an additional random string r to produce the commitment $\beta = \beta(r, x)$. We call x the content of the commitment β, and r its decommitment value. But r is often dropped in the notation, and in all remaining sections we denote a commitment to x simply as $\lceil x \rceil$.

One implementation of unconditionally hiding commitment schemes was suggested in [21]: $\beta(r, x) := h(r||x)$, where h is a hash function such as SHA256, r is a sufficiently long random string (256 bits, say), and $||$ denotes string concatenation. Though the implementation is probably sound from a practical perspective, it is not obvious how one may formally prove the hiding property of this scheme. In this respect, the result of [22] gives a much more rigorous treatment. The resulting commitment scheme is very efficient and is statistically hiding, meaning that an exponentially small amount of information is leaked. This scheme is certainly good enough for our purposes, though our claims would have to be restated since the resulting protocol would be statistically private, not unconditionally.

However, another interesting candidate exists: commitments of the form $\beta(r, x) := g_0^r g_1^x$, where g_0 and g_1 are arbitrarily chosen elements of some finite group G in which computing the discrete logarithm is assumed to be difficult, an assumption which is the basis for many cryptographic protocols. It is easy to see that these commitments (sometimes known as Pedersen commitments but introduced in [23]) are unconditionally hiding, and that they are binding, provided the Sender cannot calculate the discrete log of g_1 with respect to g_0 in G. In fact, these commitments have many interesting properties which make them very suitable for voting (see [8][14]), but this is not the focus of the current paper.

B. Entities involved in voting

For simplicity of exposition, we assume there exists one *Election Authority* (EA) which is responsible for running the election. Our protocol makes it very hard for it to alter the outcome of the election, but it can leak the privacy of individual votes. To mitigate this threat, we will occasionally describe how certain responsibilities can be shared among several authorities, all of which need to be corrupt in order to leak votes.

The EA runs an *Audit Table*, which can be implemented as a database whose contents are replicated on a public web site. This Audit Table only accumulates data, meaning that data, once published, cannot be withdrawn. This property can be ensured by using standard techniques, such as the creation of dependencies between successive alterations using cryptographic hash functions.

Voters, besides casting their votes, should make sure that somebody (either the voter herself, or some helper organization) verifies that the ballot receipt received corresponds to the image published by the Election Authority on its web site; if not, voters must complain.

The election result is certified by *Auditors*, whose role is to verify the correctness of the procedures executed by the EA. The Auditors' burden is light: they must ensure that the challenge bits used in the pre- and post election audits are unpredictable from the EA's point of view, and then perform all the checks to verify that the EA executed its tasks honestly.

An *observer* is any diligent entity with computational resources, who also verifies the election process. In fact, the tasks performed by Auditors and Observers are quite similar, except that the latter are not officially recognized by the EA and thus have no role in generating the random audit challenges.

III. Comparing the ballot layouts of PaV and PS

A. The Prêt-à-Voter ballot

The ballots used in PaV are described in detail in [5], section 4. A canonical ordering of candidates must be defined. In this section it is 0: Alice; 1: Bob; 2: Charles; 3: Diane. An example ballot looks like this:

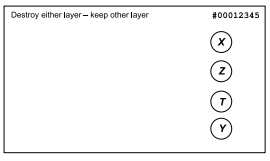
1: Bob	
3: Diane	✗
0: Alice	
2: Charles	
Destroy this part	Keep this part z4fqkr3c

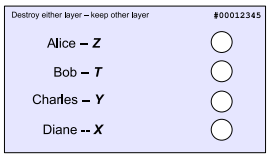
The left part contains a permutation of the candidates. The right part is empty except for the last row, and the voter votes by putting an ✗ in one of its first four cells. The magic string `z4fqkr3c` (in reality probably longer) is an encryption of the permutation used to shuffle the candidate order on the ballot, encrypted with the public keys of the mixes.

The process of ballot casting consists of separating the left and the right column, destroying the left column and scanning the right one. Either manually, or through OCR, the row containing the ✗ and the encryption of the permutation are associated with the ballot image. The voter can take the right column home as a receipt.

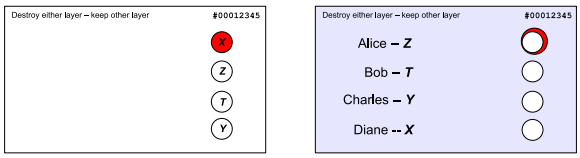
B. The PunchScan ballot

The top and bottom layers of a typical PS ballot are depicted below. In its original form, the typical PS ballot employs auxiliary symbols, such as X, Y, Z and T, which make the connection between the option chosen on the top and bottom layers, and which are different for each ballot. We could have used any set of four different symbols, or four colors etc. Observe that on both layers these auxiliary symbols are permuted. In the example ballot below, a vote for Diane corresponds to the auxiliary symbol X.



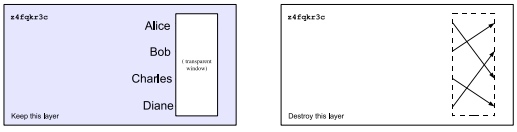


When the two layers are superimposed, a vote is cast by using a dauber that affects both layers simultaneously, leaving a mark on both layers. The voter chooses one of the two layers to take with her as a receipt, the other layer is destroyed.

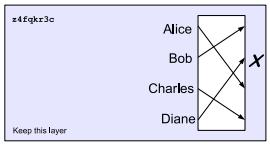


C. Differences between PaV and PS Ballots

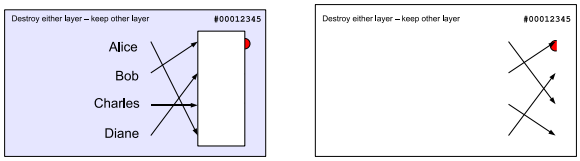
With respect to ballot layout, the differences between Prêt-à-Voter and PunchScan can be illustrated explicitly using a ballot with two layers and using arrows. In this representation, the equivalent of the Prêt-à-Voter ballot presented above then has a top layer (the former left side) with a transparent 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 for which the two layers have been separated. Note that both layers contain a part of the mark, that the voter can choose which layer she wishes to keep, and that neither layer reveals information about the vote.



In summary, the arrow representation makes it explicit 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.

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

1) As just explained, 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 on top of each other. However, PaV can be changed to resemble the PS layout.
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 information that goes through an auditing process.
5) PaV uses decryption mixing as the underlying cryptographic primitive, whereas PS uses a bit commitment scheme.

D. Similarities between PaV and PS Ballots

The purpose of the former subsection was to stress the differences between PaV and PS because of the corresponding cryptographic implications. Now we will do the opposite, and stress the similarities. In particular, it might seem that cutting the vote in two, as in PaV, severely limits its ballot layout (as argued in [16], page 117), but we demonstrate a simple trick to get around this apparent limitation. We also demonstrate how multiple races may be placed on the ballot, and how to enter a number several digits long.

The following ballot layout illustrates the mean idea; once this is clear, generalizations are straightforward.

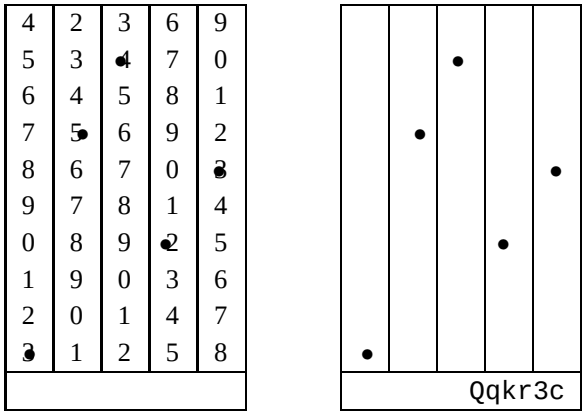
Race 1	**Race 2**	**Race 1**	**Race 2**
3: Diane	1: Einstein	✗	
0: Alice	2: Newton		
1: Bob	0: Bohr		✗
2: Charles			
(Offset $x = 1$)	*(Offset $x = 2$)*	`Qqkr3c`	`bkuryt`

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.

There are many layout options based on this idea, hence another important issue is to design ballot layouts that do not leave the voter confused. We outline one option for concreteness. Suppose that all the four columns have the same width. Imagine now a mask that shows only the first column. In addition, imagine a horizontal ruler which highlights one cell in the first column, and has a hole at the same height in the third column, through which the voter can write an ✗ or perforate a hole. By shifting the mask one column to the right

(or the ballot to the left), the second and fourth column are similarly 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 her 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 shows the upper and lower layer of a vote for candidate 35423. (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.)



Note that the result 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 all voters.

E. Oblivious Vote Capture

The fact that the bottom sheets are identical for all voters (except for the ballot id, see below) points to another solution: the bottom sheet could be generated by a voting machine, thus helping to reduce voter mistakes. The top sheet, now made of transparent material, is put on top of the screen of the voting machine. The voter is guided by the voting machine in making her selections; the voting machine enforces, for example, the requirement that each column bear at most one mark. The machine does not get to know the vote, however, as it also depends on 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 voting machine stores the information about the cells chosen, which it forwards to the bulletin board.

A small detail here has to do with the ballot id, which the voting machine must learn. One possibility is to have a barcode sticker on the (top) sheet, which will be read by the voting machine before the voter begins, and which is printed on the receipt. As one of the last steps, the sticker is peeled off the sheet and stuck just above its image printed on the printout, allowing for quick visual verification to make sure that the voting machine copied the barcode faithfully.

IV. The new protocol

A. Ballot layout and casting a vote

We will be using the Prêt-à-Voter ballot layout, but for simplicity of exposition we restrict ourselves to *cyclic* permutations. We stress, though, that our protocol can be generalized to arbitrary permutations.

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 chosen differently and randomly for each ballot. As before, a voter marks a vote by putting a ✗ on the right part of the ballot.

B. The audit table

Since Punchscan uses two permutations, 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 $x = x' + x'' \pmod m$. The Election Authority(EA) commits to x, x' and x'' publicly, and uses a unique ballot id number i to establish the link between the commitments 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, $y = x + v \pmod m$, 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. This table is very similar to that of PunchScan. Note that there are $2S$ rows, where S is an upper bound on the number of voters. Note also the commitment notation on the symbols for some columns. 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 all the voter ids, n, and the scanned image of the ballot receipts, r, will be published as well.

i	$\lceil x \rceil$	n	r	y	$\lceil \pi_1^{-1} \rceil$	$\lceil x' \rceil$	$y-x'$	$\lceil x'' \rceil$	$\lceil \pi_2 \rceil$	v
1										
⋮										
$2S$										

Observe that the table is divided into left, middle and right parts. 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 $y(i) = x(i) + v(k) \pmod m$ we are talking about the values corresponding to the *same* ballot. The

columns labeled $\lceil \pi_1^{-1} \rceil$ and $\lceil \pi_2 \rceil$ are used as commitments for these two permutations. During the audits, commitments are opened to verify that the EA did not cheat, as will be explained below.

In fact, as will become clear when we discuss the post-election audit, it is necessary to create L copies of the audit table. More precisely, we need copies of the middle and the right part of the audit table, each with the same values for x, but with different values for π_{1l}^{-1}, x'_t, x''_t and π_{2l}. Here L is a security parameter and l a subscript between 1 and L.

C. Protocol description

Phase 1: System set-up, ballot creation, pre-election audit

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

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.

1.2 For each $l \in \{1, \ldots, L\}$, the EA chooses 2 random permutations π_{1l} and π_{2l} on the set $\{1, \ldots, 2S\}$. Also, for each $i \in \{1, \ldots, 2S\}$, $j_l = \pi_{1l}(i)$ the EA chooses $x(i)$, $x'_l(j_l)$ and $x''_l(j_l)$ random such that $x(i) = x'_l(j_l) + x''_l(j_l) \pmod m$.

1.3 For each $i \in \{1, \ldots, 2S\}$, $l \in \{1, \ldots, L\}$, the EA commits to $x(i)$, $\pi_{1l}^{-1}(j_l)$, $x'_l(j_l)$, $x''_l(j_l)$ and $\pi_{2l}(j_l)$. For each i and l these commitments represent one ballot, which is printed (where $j_l = \pi_{1l}(i)$).

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

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_l in the middle part of the table if $j_l = \pi_{1l}(i)$ and $i \in A$.

1.6 The Auditors check that the ballots and the entries in the audit table were created honestly, i.e. that $\pi_{1l}^{-1}(j_l)$ and $\pi_{2l}(j_l)$ are consistent with permutations (no two-to-one mappings), and that $x(i) = x'_l(j_l) + x''_l(j_l) \pmod m$ holds for each row opened, and that the serial number i and the offset $x(i)$ printed on the ballot correspond to the entries in the table.

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

Phase 2: The election

The voter acts as follows:

2.1 The voter identifies herself using existing mechanisms.

2.2 The voter chooses a ballot $i \in E$, enters the booth and marks her choice by putting an ✗, as already described in Section IV-A. She separates the two parts of her ballot and ensures that the left part is physically destroyed.

2.3 The voter leaves the booth; her receipt, the right side, is scanned. Then the row number of her mark, $y(i)$, is interpreted (by OCR or manually) which superimposes an arrow or other symbol on the scanned image. The voter confirms this interpretation, hence casting her vote. The scanned image is stored in order to be published on the Audit Table, as described in Step 3.1. The voter takes the print-out of the scanned image with arrow with her as a receipt.

Phase 3: Publishing the results

3.1 After the election, the EA publishes the voter's id $n(i)$, the scanned receipt $r(i)$ and $y(i)$ for each $i \in E$, $[y - x']_l(j_l)$ for each $j_l \in \pi_{1l}(E)$, and $v_l(k_l)$ for each $k_l \in \pi_{2l}(\pi_{1l}(E))$.

3.2 From the column labelled v the EA calculates the tally. The tally should be equal for all the L copies of the Audit Table.

Phase 4: The post-election audit

The EA could try to cheat by changing the values in column v, so an audit is necessary. We run L versions of the audit protocol in parallel, each with a different Left/Right choice. Then the probability of EA getting away is 2^{-L}. This is a well-known technique also used by Punchscan ([7], section 5.4).

4.1 The Auditors create L random bits $b_1 \ldots b_L$: Left or Right.

4.2 For $l \in \{1, \ldots, L\}$:
If b_l = Left, then for each j the EA opens $i := \lceil \pi_{1l}^{-1}(j) \rceil$ and $\lceil x'_l(j) \rceil$, and it is verified whether $y_i - x'_l(j) = [y - x']_l(j)$ holds.
If b_l = Right, then for each j the EA opens $k := \lceil x''_l(j) \rceil$ and it is verified whether $[y - x']_l(j) = x''_l(j) + v_l(k)$ holds. Observe that this equation should be satisfied because for the same ballot $y = x + v = x' + x'' + v$, so $y - x' = x'' + v$.

4.3 After the last audit procedure is completed, all the decommitment values of unopened bit commitment values are destroyed.

V. Properties of our protocol

A. Assumptions

We now state the assumptions needed to obtain a secure protocol.

A1 *The Election Authority uses a commitment scheme that is unconditionally hiding and computationally binding.*

A2 *The computational assumption of the computationally binding commitment scheme is valid until the election ends. In particular, the Election Authority cannot break the commitment schemes used.*

A3 *The ballot receipts given to voters are unforgeable and undeniable.* This can be satisfied using special paper (with watermarks etc) to print the ballot receipts.

A4 *Except for the ballot image and the information committed to in the audit table, no other copies of (the information on) the ballots exist.* In practice, this assumption may be the hardest to fulfill. It can be satisfied by putting the ballots in opaque envelopes, for instance.

A5 *The challenge bits of the Auditors are unpredictable from the EA's point of view.* This assumption can be provably assured by using standard techniques such as a beacon, or applying a cryptographic hash function to data that the EA does not control, etc.

A6 *Though the Election Authority may try to cheat in other ways, we assume that it is in its interest to collaborate until the protocol has completed.* This is reasonable in view of the fact that in many situations even corrupt EAs have an interest in *appearing* honest, and not completing the protocol is too obvious to get away with.

B. Claims

We now formulate the two principal properties of our protocol.

CLAIM 1 (PRIVACY OF THE BALLOT)
If Assumptions A1, A3 and A4 are satisfied, then the protocol presented in Section IV-C has the following property:
a) When the voter leaves the booth, the receipt that the voter retains leaks no information (in the Shannon sense) about her vote.
b) The information that the Election Authority publishes in the Audit Table leaks no information about any individual vote.
c) The additional data revealed during the post-election audit reveals no information about any individual vote.
This implies that our protocol guarantees the privacy of the voter unconditionally.

PROOF: (a): When voter i leaves the booth, her receipt contains a mark in row $y(i)$ and the serial number of the ballot, i. Since $y(i) = x(i) + v(k) \pmod m$ and $x(i)$ is chosen at random, no information is revealed as long as no information about $x(i)$ is revealed. In (a) no other information which depends on $x(i)$ is available.
(b) In addition to the image of the receipt, the Audit Table contains a commitment of $x(i)$. But since an unconditional bit commitment scheme is used, no information about $x(i)$ leaks this way.
(c) During the post-election audit, for each l either x'_l and π_{1l}^{-1}, or x''_l and π_{2l} are opened,. Since these values are chosen randomly to satisfy the equation $x(i) = x'_l(j) + x''_l(j) \pmod m$, knowing only one of them does not give information about $x(i)$. And since either π_{1l}^{-1} or π_{2l} is opened, but *never* both, none of the permutations π_l that directly link the left and the right part of the audit table is ever revealed.

CLAIM 2 (CORRECTNESS VOTE COUNT)
If Assumption A2 and A5 are satisfied, then the protocol presented in Section IV-C has the following property:
a) For each ballot that was constructed and printed dishonestly during Phase 1, the Election Authority has a probability of at least $\frac{1}{2}$ of being caught.
b) If the Election Authority did not cheat during Phase 1 but did during Phase 3, it will be caught with probability at least $1 - 2^{-L}$.

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 without being detected, the EA must cheat on x'_l and π_{1l}^{-1}, or on x''_l and π_{2l}, for *each* of the L copies of the audit table. So the EA only gets away with cheating if it successfully predicts *all* the coin flips in Step 4.1 which, under assumption A5, only happens with $p = 2^{-L}$.

Note that the last claim is true independent of whether the EA is actually honest or not, or whether the election software is correct or not. As long as Assumptions A2 and A5 are satisfied, and if the EA passes the pre- and post-election audits successfully, then, with overwhelming probability, the result published by the EA is the correct one.

C. Other requirements for fair elections

Though the previous subsection captures the essential properties of our protocol, we want to make sure that we have not overlooked any requirement for a fair voting protocol. We therefore check the list of requirements from [24], 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 cooperation 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) obtains an undeniable proof that her vote is included in the set of votes tallied.*

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

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

Assumption A3 implies that if a voter does not find her receipt published in the audit table, she has an undeniable proof that the EA is trying to cheat. So if we assume that the EA publishes all receipts, the voter can check her value y on the 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 not be possible (i) to see the vote on any ballot deposited in the ballot box; (ii) to modify a ballot; (iii) to remove a ballot from the ballot box; (iv) to add ballots not coming from voters.*

Subrequirement (i) follows from part (a) of Property 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 from the proofs of the previous requirement, E, given that a bulletin board is used.

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 election process is robust against malicious and random faults, and has a very high probability to terminate successfully.*

The only way the process does not terminate successfully is if the EA cannot open the necessary bit commitments, or if it refuses to do so. The latter contradicts Assumption A6. In any case, the impact of both types of faults can be mitigated by the use of secret sharing, as explained in the next section.

D. Secret sharing of decommitment values

Probably the most serious drawback of the protocol presented in this paper is the fact that the decommitment values of the bit commitments are in the hand of one entity: the Election Authority. Moreover, these values have to be stored until after the election, since they are needed to determine the outcome of the election, as well as to complete the post-election audit. If the EA reveals these decommitment values, then the privacy of the corresponding vote is completely compromised, so this could be used to coerce the voter. On the other hand, the EA could refuse to open them, thus preventing the tallying of the votes.

One obvious solution is to split the decommitment values over a number of different authorities, of which a certain quorum is necessary in order to reconstruct the original decommitment values. A verifiable secret sharing scheme such as the one proposed by Pedersen [25] is suitable for this task. This alleviates the problem of concentrating power in one entity, and also adds redundancy in the case of random faults.

The use of verifiable secret sharing also solves another problem. After the protocol has been completed, all the decommitment values of unopened commitments, as specified in Step 4.3, must be destroyed permanently. If a verifiable secret sharing scheme is used, then this property holds if the owners of the shares act honestly by *refusing* to further open decommitment values and destroying permanently any data related to their shares.

VI. PROTOCOL VARIATIONS

A. Alternative audit procedures

The audit procedures described in Section 4 are among the simplest so that the protocol properties hold, but some additional comments are appropriate.

In the first place, all unused ballots and corresponding commitments can and should be opened and audited after the election. There is no benefit to not doing so, and the audit increases the chances of catching a dishonest EA.

In addition, other auditing protocols could be used. Apart from the approach used here, PunchScan presents another (standard) approach ([6] section 7.6, [7] section 5). It is the equivalent of one round of randomized partial checking as presented in [21]. Unlike the mechanism presented in Step 4, there is only one copy of the table, and for each j in the middle part of the table a random challenge bit is created: *Left* or *Right*, which has the following semantics:

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

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

Using this approach we catch a cheating EA with probability $\frac{1}{2}$ 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 the 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 the number of voters is small.

One variation to avoid this is the following: instead of using two permutations, we use four. We also split x into four parts: $x = x_1 + x_2 + x_3 + x_4 \pmod m$ and use two rounds of randomized partial checking. The audit table now has five parts and essentially looks like this:

$\lceil x \rceil$ y	$\lceil x_1 \rceil$ $y{-}x_1$ $\lceil x_2 \rceil$	$y{-}x_1{-}x_2$	$\lceil x_3 \rceil$ $x_4{+}v$ $\lceil x_4 \rceil$	v

Observe that $y - x_1 - x_2 = x_3 + x_4 + v \pmod m$. 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 [21], 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 [26] it is proven that a constant number C of auxiliary permutations is sufficient to get it below any ϵ, but an explicit estimate of which value for C should be used in practice is not given. This is subject of current research.

B. A brief description of Punchscan

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

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

The first row shows the notation introduced in this paper, whereas the second row shows the notation of [7] and [6]. 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 $1 \pmod 2$ 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 $p = \frac{1}{2}$. Therefore x_1, x_2, t_1 and t_2 are chosen randomly satisfying $x_1 + x_2 = t_1 + t_2 \pmod m$.

C. A simpler but less efficient protocol

We can obtain a less efficient, but a conceptually simpler protocol by using a commitment scheme that allows the Sender to show linear relations between two or more commitments modulo an integer m. For instance, suppose the Sender would like to show that $\lceil x_1 \rceil + \lceil x_2 \rceil \equiv 0 \pmod m$. That is, he wants to show that the equality $x_1 + x_2 \equiv 0 \pmod m$ holds without revealing any other information about x_1 or x_2.

For $m = 2$ there exists a general construction to accomplish this property for *any* kind of bit commitment, at the expense of a factor $2L$, where L is a security parameter. The idea, attributed to Bennett and Rudich, is described in Section 2.2 of [27], where it is called "Bit Commitment with XOR", denoted as $\overrightarrow{x}$. The idea is to represent each commitment as a vector of pairs of simple bit commitments, such that each pair xors to the committed bit value. See [27] and [13] for more details. It is easy to see that this scheme generalizes to proving linear relations between many commitments, and that xor operations on bits can be generalized to addition on integers modulo m.

Let $\overrightarrow{w}$ be a copy of $\overrightarrow{x}$, i.e. $w = x$. The audit table of our variant consists of four parts, which are related through three permutations, π_1, π_2 and π_3.

$\overrightarrow{x}$	y	$\overrightarrow{w}$	$\overrightarrow{y}$	$\overrightarrow{v}$	v

We now apply the customary cut-and-choose approach: the EA has two choices, depending on the challenge. Either it opens π_1 and π_3 and shows that $\langle \overrightarrow{x}, y \rangle$ corresponds to $\langle \overrightarrow{w}, \overrightarrow{y} \rangle$ according to π_1 and that $\overrightarrow{v}$ corresponds to v according to π_3. Or the EA opens π_2 and proves that $\overrightarrow{w} + \overrightarrow{y}$ modulo m corresponds to $\overrightarrow{v}$ according to π_2, using the Bennett-Rudich techniques explained above.

VII. Efficiency considerations

A concern about our protocol might be that it involves a large number of commitments, and since each commitment requires a decommitment value of at least 200 bits, it might seem that the sheer amount of data becomes too large to make the protocol work in practice. Instead of analyzing exactly how many commitments would be needed, we show that our protocol has a special property, which allows us to replace any number of *bit* commitments to a constant number of *string* commitments, where this constant is linear in the security parameter, L.

More precisely, in [28] the following is shown. Suppose that we have a three-step protocol with the following structure. In Step 1, the prover commits to a large collection of bit commitments. In Step 2, the verifier issues a random challenge. Finally, in Step 3, based on the commitments and the challenge, the Sender opens a certain subset of the bit commitments. Then this protocol can be converted into a protocol that used two string commitments in each round, and which reduces the probability to catch a cheating Sender in each round by a factor of two. This technique applies, except that we have a pre-election and post-election audit, meaning that we have an additional Step 4, in which the Receiver issues another challenge, and Step 5, in which the Sender opens a second subset.

Let us describe how to adapt this technique to our situation, i.e. how to convert an arbitrary large collection of commitments into string commitments: the sequence of bit commitments $b_1, \ldots b_i$ is split into $B_1 := b_{11}, \ldots b_{1i}$, $B_2 := b_{21}, \ldots b_{2i}$, and $B_3 := b_{31}, \ldots b_{3i}$ such that $b_j = b_{1j} \oplus b_{2j} \oplus b_{3j}$ holds, and two of the three values are chosen at random. Then the Sender commits to B_1, B_2 and B_3, using string commitment.

In order to open the first subset, the parties proceed as follows: for each b_j that must be opened in the original protocol, the Sender announces the value b_{1j}, b_{2j} and b_{3j}. The verifier then issues a random challenge, telling the Sender either to open B_1, B_2 or B_3, and, after receiving the decommitment values, verifies whether the Sender opened honestly. In order to open the second subset in the post-election audit, the same procedure is repeated, except that the challenge now refers to the two remaining unopened strings.

Using this technique decreases the probability to catch a cheating Sender in one round by a factor of three. This means

that if initially we had a $\frac{1}{2}$ probability of catching a cheating authority, this probability has now reduced to $\frac{1}{6}$. So in order to catch a cheating authority with probability $1 - 10^{-9}$, we need at least 114 (the smallest integer n such that $(\frac{5}{6})^n < 10^{-9}$) rounds, and three times the number of string commitments. In other words, $3 \times 114 = 342$ string commitments are sufficient to commit to the whole Audit Table. So we can conclude that storage and communication requirements for our protocol are truly modest.

VIII. Conclusion

This research began as a comparison of Prêt-à-Voter and PunchScan. Surprisingly, we found a third variation which is an improvement on both: the simpler ballot layout of Prêt-à-Voter with the simpler cryptography of PunchScan. As our analysis shows, the three protocols are very similar and can be considered variations on the same theme.

We believe that the two most important properties of the protocol presented here are the fact that it provides unconditional privacy, and that it is very simple. The latter property is important in election systems in order to convince a wider audience of protocol soundness. An additional advantage is that the protocol's security, which is already quite intuitive, should be relatively easy to model and prove more rigorously, for instance in the universally composable model. This is the subject of current research.

Another open question is how to design a voting system if the number of candidates is too large to fit on the ballot, as is the case in Brazil. Though we have indicated a possible solution, we believe more study is necessary.

Acknowledgment

The author would like to thank Peter Ryan, Ben Hosp and Stefan Popovenuic for help while writing this paper. Special thanks are due to Poorvi Vora, whose proofreading of the paper improved the final presentation considerably.

References

[1] J. Benaloh and D. Tuinstra, "Receipt-free secret-ballot elections," in *STOC '94 — 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] P. Y. A. Ryan, "A variant of the Chaum voter-verifiable scheme," in *WITS 2005*, pp. 81–88, ACM, 2005. Also published as Tech. Rep. CS-TR 864, University of Newcastle, October 2004.

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

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

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

[8] R. Cramer, M. K. Franklin, B. Schoenmakers, and M. Yung, "Multi-autority secret-ballot elections with linear work," in *EUROCRYPT '96*, LNCS 1070, pp. 72–83, 1996.

[9] T. Moran and M. Naor, "Receipt-Free Universally-Verifiable Voting with Everlasting Privacy," in *CRYPTO 2006*, LNCS 4117, pp. 373–392, 2006.

[10] T. Moran and M. Naor, "Split-ballot voting: everlasting privacy with distributed trust," in *CCS 2007*, pp. 246–255, ACM, 2007.

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

[12] J. Bos, *Practical Privacy*. PhD thesis, Technische Universiteit Eindhoven, 1992. Available online: http://alexandria.tue.nl/extra3/-proefschrift/PRF8A/9201032.pdf.

[13] J. van de Graaf, "Anonymous one-time broadcast using non-interactive dining cryptographer nets with applications to voting," in *Anais do VII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pp. 68–79, Sociedade Brasileira de Computação, 2007.

[14] J. van de Graaf, "A note on unconditionally private voting protocols based on discrete log commitments." In preparation, 2009.

[15] B. Adida and C. A. Neff, "Ballot casting assurance," 2006. Available online: http://www.usenix.org/events/evt06/tech/full_papers/adida/adida.pdf.

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

[17] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. T. Sherman, and P. L. Vora, "Scantegrity: End-to-end voter-verifiable optical-scan voting," *IEEE Security & Privacy*, vol. 6, no. 3, pp. 40–46, 2008.

[18] B. Adida and R. L. Rivest, "Scratch & Vote: self-contained paper-based cryptographic voting," in *WPES 2006 — ACM Workshop on Privacy in the Electronic Society*, pp. 29–40, ACM, 2006.

[19] S. Popoveniuc and P. L. Vora, "A framework for secure electronic voting." WOTE 2008 — IAVoSS Workshop On Trustworthy Elections, Leuven, Belgium, 2008. Accepted for publication in Cryptologia.

[20] C. Crépeau, "Commitment," in *Encyclopedia of Cryptography and Security*, Springer, 2005.

[21] M. Jakobsson, A. Juels, and R. L. Rivest, "Making mix nets robust for electronic voting by randomized partial checking," in *USENIX 2002*, pp. 339–353, 2002.

[22] I. Damgård, T. P. Pedersen, and B. Pfitzmann, "On the existence of statistically hiding bit commitment schemes and fail-stop signatures," *J. Cryptology*, vol. 10, no. 3, pp. 163–194, 1997.

[23] D. Chaum, I. Damgård, and J. van de Graaf, "Multiparty Computations Ensuring Privacy of Each Party's Input and Correctness of the Result," in *CRYPTO '87*, LNCS 293, pp. 87–119, 1987.

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

[25] T. P. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," in *CRYPTO '91*, LNCS 576, pp. 129–140, 1991.

[26] 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, 2003.

[27] C. Crépeau, J. van de Graaf, and A. Tapp, "Committed Oblivious Transfer and Private Multi-Party Computation," in *CRYPTO '95*, LNCS 963, pp. 110–123, 1995.

[28] J. Kilian, S. Micali, and R. Ostrovsky, "Minimum resource zero-knowledge proofs (extended abstract)," in *CRYPTO '89*, LNCS 435, pp. 545–546, 1989.

Jeroen van de Graaf was born in Amsterdam, the Netherlands in 1960. He received a Master Degree in Mathematics from the Universiteit van Amsterdam in 1985. Subsequently he worked in the Cryptography Group at the CWI in Amsterdam. After working for six years in industry, he started a Ph.D. on quantum cryptography at the Université de Montréal in Québec; receiving the degree in 1998. Currently he is an Assistant Professor at the Departamento de Computação of the Universidade Federal de Ouro Preto, Brazil. His research interests include cryptographic protocols, in particular voting, and protocols aimed at preserving the privacy of individuals. Contact him at research@jvdgraaf.net.