Adapting Chaum's Voter-Verifiable election scheme to the Brazilian system

Jeroen van de Graaf

[1]Laboratório de Computação Científica – Universidade Federal de Minas Gerais
Avenida Antônio Carlos 6627 – 31270-901 Belo Horizonte (Pampulha) – MG

jvdg@lcc.ufmg.br

Abstract. *Chaum's novel election scheme allows for the voter to obtain a receipt through which he can verify that his vote is included in the tallies without revealing any information about the vote cast. Chaum's original scheme uses visual cryptography, and it is necessary to have a printer able to print on both side simultaneously. Here we simplify this assumption requiring that the printer be able to erase completely certain areas of the ballot by overprinting. We show which ballot layouts could be used for this purpose, and give a formal description of the simplified protocol.*

Resumo. *O novo esquema eleitoral do Chaum permite que o eleitor obtenha um recibo através do qual ele pode verificar se seu voto foi incluído na apuração, sem revelar qualquer informação sobre seu voto. O esquema original do Chaum é baseado no criptografia visual e requer uma impressora que imprime a dois lados simultaneamente. Aqui simplificamos esta requisição, exigindo que a impressora seja capaz de apagar completamente certas áreas da cédula sobreescrevendo-as. Mostramos quais* lay-outs *podem ser usados, e damos uma descrição formal do protocolo simplificado.*

1. Introduction

Giving voters a receipt of their ballot seems impossible: either you violate voter privacy, or the receipt is something meaningless. In [2] Chaum presents a system which overcomes this paradox. In a nutshell, his idea is as follows: the vote is presented to the user as a combination of two parts (top and bottom layer in the original scheme, left and right side in this paper) such that *1)* together the two parts show the vote to the voters; *2)* individually each part does not reveal any information about the vote; *3)* the voter must choose one part which he will take home; the other part will be destroyed.

The scheme uses some sophisticated cryptographic protocols. In particular, it is based on the notion of *mix networks* or *mixes*. The basic idea is that there exist an authority who accepts a batch of encrypted votes. Each authority commits itself to take such a batch, permute (or mix—hence the name) the votes in it, decrypt each vote and publish the result. There exists a protocol that can be used to verify whether the authority is mixing

honestly (and a cheating authority would be caught with a high probability). By putting several authorities in sequence, voter privacy is guaranteed as long as at least one of them is honest. See [5] for details.

In Chaum's scheme the mix network is used as follows: the election machine knows which part the user has chosen and passes it on to the tallying authority, who publishes the representation of the ballot on the internet. Since the voter has his receipt, he can verify that his vote is present on the internet (if absent he has a proof that the tallying authority is cheating). Associated to his ballot is an encrypted version of his vote, and he can verify that this version is indeed fed to the mix network. Since it is presumably impossible to cheat the mixing process, the voter can trust that his vote will come out of the mix network and will be included in the tally.

In this paper we will focus on adapting Chaum's initial scheme to the Brazilian election system. More precisely, Chaum's original idea is based on visual cryptography and on printers that can print on both sides of a piece of paper simultaneously. Here we simplify this assumption: we require the existence of printers that can overprint some areas of the ballot in such a way that it becomes impossible to deduce what was written before. Note that in particular for thermal printer this is very reasonable: by exposing the same area to considerable heat for a second time, it is possible to blacken out completely what was written.

It is important to mention that, though elegant, there is no real *need* for visual cryptography in Chaum's scheme. It does allow for arbitrary pixel representations of voting alternatives, but often simpler ballot encodings are possible, like the ones presented here. The only requirements for the system to work are the three properties mentioned in the first paragraph of this paper. Whether using visual cryptography or not, the remainder of the cryptographic protocol remains essentially the same. However, the resulting protocol without visual cryptography is considerably simpler, and therefore easier to explain to voters and easier to analyze, not mentioning the technical challenges of double-sided printing.

The contributions of this paper are two-fold: first, in Sections 2 and 3, we show new ballot lay-outs suited to the Brazilian election system. One version was part of a prototype briefly shown at SSI 2003, but the most recent lay-out is new. Second. in Sections 4 and 5, we give a more formal description of the protocol. Since the protocol here is simpler than the one presented in [2], and since the formal description there is rather dense, we believe that our description will help in understanding the cryptographic characteristics of the scheme.

The mixing process is not subject of this paper. For details see [3] and [5]. A detailed description of Chaum's protocol can be found in [1] and [4], research that was done concurrent with ours.

2. Ballot lay-out overprinting in stripes

To simplify the discussion, suppose that the voter has to choose between 10 options, labeled 1, 2, ..., 0. We assume that the equipment creates a receipt by using two symbols,

for instance "<" and ">". We adopt the following rules:

- the system chooses the left symbol at random;
- for the options that are NOT the voter's choice, the system produces the other symbol, thus producing <> or ><;
- for the voter's choice the system uses the same symbols, << or >>.

Printing could be done vertically or horizontally. We choose the latter here to save space (but see the discussion about cutting the ballot in two halves at the end of this section):

```
1  2  3  4  5  6  7  8  9  0
<> <> >< <> >> >< <> <> <> ><
```

In this example, the voter can verify that his vote is for alternative 5, since it is the only alternative in which the left and the right symbol are equal.

After the voter has verified that his choice has >> or <<, he must choose between *Right*, meaning *preserve right* side symbols and overprint left side symbols, or *Left*, with the obvious opposite semantics. Depending on this choice, the printer will blacken out the left symbol or the right symbol of each pair (we are writing the blackened out position with a dot):

```
1  2  3  4  5  6  7  8  9  0
.> .> .< .> .> .< .> .> .> .<
```

or

```
1  2  3  4  5  6  7  8  9  0
<. <. >. <. >. >. <. <. <. >.
```

Note that these two resulting images individually reveal no information about the vote, because the symbols not blackened out are based on random bits. The information about the voter's choice is in the **combination** of the two images, but we are assuming that one of them will be destroyed. This situation is very similar to the One Time Pad, in which both the key and the cryptogram individually are indistinguishable from random bits or characters, but where knowing both reveals the message. Summarizing:

- the symbols printed are completely random;
- the information about the vote is not in the symbols, but in whether the left and the right symbol of each pair are different or equal;
- after the voter's verification, either all left symbols of a pair or all right symbols are destroyed, meaning that the remaining symbol has a 50-50 percent probability distribution and hence reveals no information;
- the system cannot know beforehand which side will be chosen; this is essential to the security of the scheme.

Once you get the basic idea, there are endless variations: instead of <>, use () or [], triangles, arrows, etc. See the upper part of Figure 1.

One can also invert the rule: the system prints the second symbol equal to the first symbol **unless** it is the voter's choice:

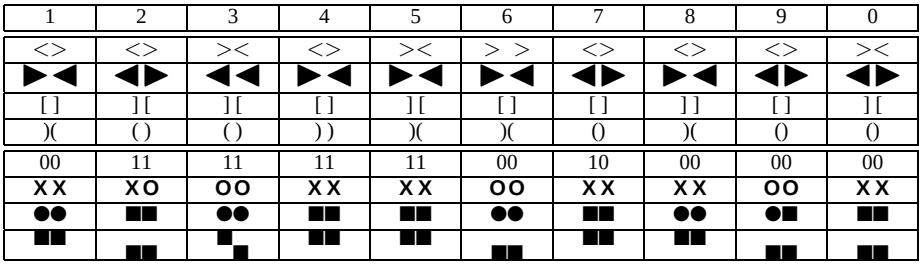

Figure 1: Possible encodings of the voter's choice. The options chosen are: 6,3,8,4;7,2,9,3.

```
1  2  3  4  5  6  7  8  9  0
++ ++ oo ++ +o ++ oo oo oo ++
```

In this case, basically any pair of symbols goes: 01 (somewhat silly but possible), OX, +X , squares printed at different distance relative to the baseline, etc etc. Note that all the codings, though slightly different, use the same idea: the voter's choice is where there is an exception, an irregularity. Psychological tests could decide which format is the most suitable for fast recognition by voters and reduces errors (misinterpretations). See the second half of Figure 1.

In this section, the essential property expected from the printer is the ability to print black vertical bars over the full length of the vote, making it impossible to determine what the text printed earlier was. We call the overprinted image a *mask*. So here we are considering a mask of vertical stripes, who has only two positions: left or right. A possible hardware implementation could be to have a second printer head at a suitable distance from the first, whose only task is to print these stripes. By allowing to shift this head physically to the left or to the right, the overprinting could be accomplished.

Figure 2 shows the implementation of a full ballot using this idea. During elections in Brazil, each candidate is assigned a number, the size of which depends on the office. We use an imaginary election of *deputado federal* (5 digits), *governador* (2 digits), *presidente* (2 digits). In this example, the ballot encodes one vote for a candidate for the office of *Deputado Federal* whose number is 51233, candidate 72 for the office of *Governador*, and candidate 38 for the office of *Presidente*. From the example it should be clear how this scheme can be generalized.

Figure 2a shows what the voter sees when he verifies the vote. When he chooses *Right* all the left symbols of each pair are blackened out, and the receipt he takes with him looks as depicted in Figure 2b. When the voter chooses *Left* all the right symbols of each pair are blackened out, as is shown in Figure 2c.

In this paper we focus on overprinting, but the reader might wonder whether cutting the ballot in two halves and destroying one halve would do the job. In principle it would work, but in practise the ballot lay-out becomes very awkward, resulting in very long strips of paper, on which verifying the vote becomes difficult. And provided that the overwriting process erases perfectly, the method presented here provides a proof to the voter that the other halve is indeed destroyed.

1	2	3	4	5	6	7	8	9	0
Deputado Federal									
O O	O O	X X	O O	X O	O O	X X	O O	O O	X X
O X	X X	X X	O O	O O	X X	O O	X X	O O	X X
O O	O X	X X	X X	X X	O O	X X	O O	O O	X X
O O	O O	X O	O O	X X	O O	X X	O O	O O	X X
O O	O O	O X	O O	X X	X O	X X	O O	O O	X X
Governador									
O O	X X	O O	X X	X X	O O	X O	X X	O O	X X
X X	X O	O O	X X	X X	O O	X X	X X	O O	X X
Presidente									
X X	O O	O X	X X	X X	X X	O O	X X	X X	O O
O O	O O	X X	X X	X X	O O	X X	O X	O O	X X

(a) The ballot as the voter sees it (see the text for an explanation).

1	2	3	4	5	6	7	8	9	0
O	O	X	O	O	O	X	O	O	X
X	X	X	O	O	X	O	X	O	X
O	X	X	X	X	O	X	O	O	X
O	O	O	O	X	O	X	O	O	X
O	O	X	O	X	O	X	O	O	X
O	X	O	X	X	O	O	X	O	X
X	O	O	X	X	O	X	X	O	X
X	O	X	X	X	X	O	X	X	O
O	O	X	X	X	O	X	X	O	X
1	2	3	4	5	6	7	8	9	0

(b) The ballot after the voter has chosen *Right*.

1	2	3	4	5	6	7	8	9	0
O	O	X	O	X	O	X	O	O	X
O	X	X	O	O	X	O	X	O	X
O	O	X	X	X	O	X	O	O	X
O	O	X	O	X	O	X	O	O	X
O	O	O	O	X	X	X	O	O	X
O	X	O	X	X	O	X	X	O	X
X	X	O	X	X	O	X	X	O	X
X	O	O	X	X	X	O	X	X	O
O	O	X	X	X	O	X	O	O	X
1	2	3	4	5	6	7	8	9	0

(c) The ballot after the voter has chosen *Left*.

Figure 2: A real ballot layout.

3. Ballot lay-out with overprinting of arbitrary areas

Instead of using a second printer head responsible for printing the mask, another hardware implementation is possible: the paper is rewound, back into the printer and the same printer head is used to overprint parts of the original text. This makes it possible to consider more sophisticated ballot layouts, in which masks overprinting arbitrary areas (not just columns) can be employed. In particular, it becomes interesting to investigate the possibility of implementing real digits, using the well-known seven segment representation.

For many months we thought this was impossible, but a key (in retrospect, straightforward) insight is that one can encode each segment similar to the encoding explained in section 2 of this paper. Using the ox notation for each segment, one obtains

```
  o o         . .
 o   x       #   #
 x   o       #   #
  x o         # #
 x   o       .   #
 x   x       .   #
  o o         . .
```

as a possible encoding for the digit 4. Here oo or xx mean presence of a segment, whereas ox or xo means absence.

Observe that, since there are seven segments, there exists $2^7 = 128$ possible encodings for 4: by choosing the left or the upper symbol of each segment, the right or lower symbol is fixed. There are ten digits, meaning that of all the 2^{14} theoretical encodings only 1280 correspond to a valid digit.

When it comes to letting the voter choose between two alternatives, the only requirement is that of each segment only one symbol will be chosen. There are many ways in which this can be accomplished, but offering the following two choices seems very natural:

```
  . x         o .
 .   x       o   .
 o   .       .   x
  x .         . x
 x   .       .   o
 .   o       o   .
  . x         o .
```

Of course, the encoding just presented is too crude for practical use, since it is almost impossible to recognize visually the digits in this representation. A first approach is to play with the segment ends: absence of a segment corresponds to a continuous line, whereas presence means a flip of the line ends. However, this affects the readability of the digits negatively; see figure 3.

A better approach is to work in the middle of the segments, since it allows to make the artwork at the segment ends more sophisticated. Absence of a segment corresponds to a

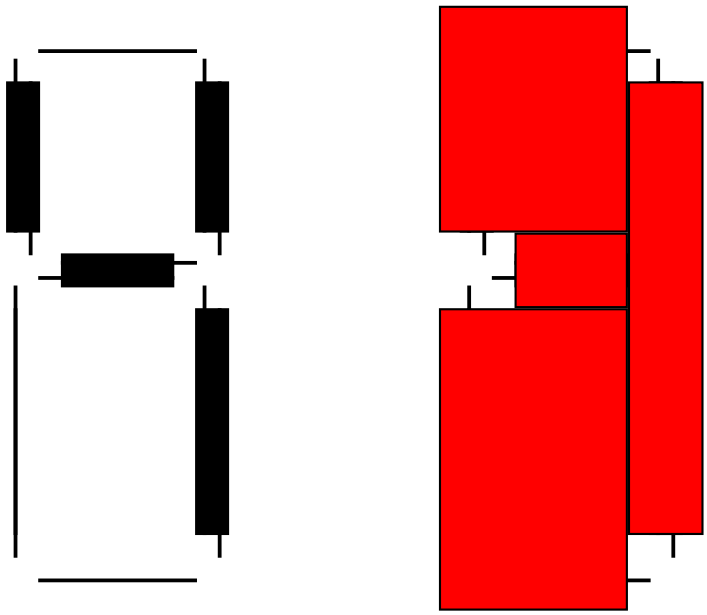

Figure 3: A sub-optimal representation of the digits, using 4 as an example

continuous line, whereas presence corresponds with a discontinuity in the line end. Figure 4 shows the digit 4, first the way the user sees, and second what the voter keeps after any area that could reveal the vote have been overprinted. It shows what happens when the user chooses *Right*. If the user chooses *Left*, its corresponding mask has vertical symmetry.

4. A formal description of the system

4.1. An example ballot

To describe the system formally, we will start by explaining how a ballot looks. Consider a simplified situation in which there are only three candidates, labeled 1, 2 and 3. We call these the *voting options* or just *options*.

A ballot contains all possible voting options, but in permuted order. The permutation is only known to the part of the system responsible for creating the votes, not to the mixes. The permutation in the example below is $\texttt{Perm}([123]) = [312]$.

A ballot also has a serial number and some encrypted stuff, which we will explain below. Using the `xo` encoding presented in the Section 2 we get something that looks like this:

```
nr=79572618;  oo ox xx;  EncryptedStuff(oo ox xx,2);
nr=79572618;  xx xx xo;  EncryptedStuff(xx xx xo,3);
nr=79572618;  xo xx oo;  EncryptedStuff(xo xx oo,1);
```

Once created, a ballot can have only two possible destinies: either the ballot creation authorities will be required to open the ballot completely and show it was constructed honestly, or the ballot will be used on election day by some voter. Since this decision will be ruled by a random process out of control of the ballot authorities, they run a very high risk of being caught when cheating.

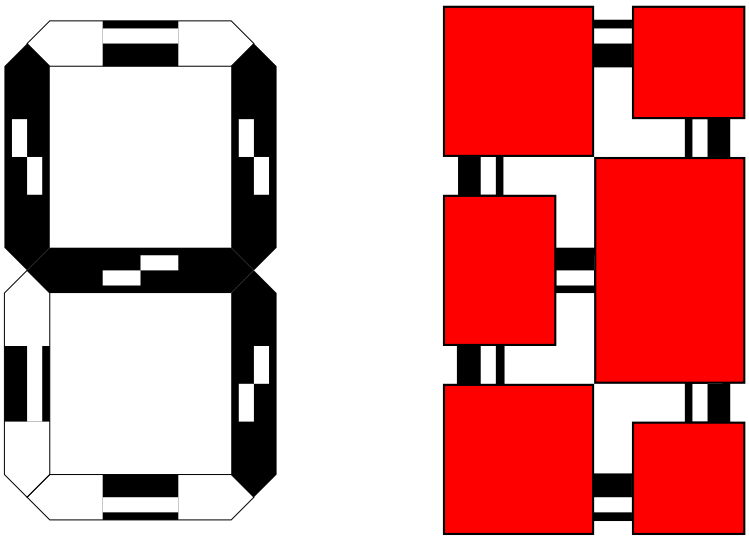

Figure 4: A better representation of the digit 4.

The `EncryptedStuff` basically consists of two parts, corresponding to what the user would see if he chooses *Left* or *Right*. For instance, suppose the voter chooses candidate 2, that is, the first line of the ballot, and he chooses *Left*. In that case, the system must guarantee that the voter can verify his ballot image `O.O.X.` on the internet, and that he can check that the `EncryptedStuff` is fed as input to the mix. The mix protocol guarantees that the encrypted vote for candidate 2 comes out of the mix. We have therefore two components, written as `VoteReprLeft`(2) and `DbleMix`(2). Likewise, there must be a `VoteReprRight`(2) and `DbleMix`(2).

However, since the combination of `VoteReprLeft`(2) and `VoteReprRight`(2) reveals the value of the voting option, it is necessary to add an additional layer of encryption, which we call `Commit(.)`.

It is very important to realize that all these functions use random strings as additional input, called random strings, keys or padding. In the notation above they are absent; if present we will put them between $\{\}$. For instance, encryption would be written as `Encr`$\{K\}(M)$.

4.2. A more general description

Let us try to describe a ballot in its most general form. Each ballot consists of T voting options (in the example above $T = 3$). The first voting option, that is, the first row, consists of

```
Nr; VoteRepr(2); Commit(VoteReprLeft(2), DbleMix(2));
                 Commit(VoteReprRight(2), DbleMix(2));
```

where `Nr` is the serial number, `VoteRepr(.)` is representation of the vote, `Commit(.)` is string commitment, `VoteReprLeft(.)` is the left side of the vote representation while the right side is erased, `VoteReprRight(.)` likewise with left and right interchanged, and `DbleMix(.)` is the component that will be sent through two mixes, hold by one authority. Let us discuss these functions in more detail:

VoteRepr{r}(v) In the vote representation used here, a vote for option 3 is represented as $r_1 r_1\, r_2 r_2\, r_3 \overline{r_3}$, where r_1, r_2 and r_3 are random bits, and $\overline{r_3}$ is the bit's complement: $\overline{r_3} = r_3$ XOR 1. For example, if $r = r_1 r_2 r_3 = 101$, then $\texttt{VoteRepr}\{101\}(3) = 110010$ (disagreement in the third pair). When using the `xo` coding, then we assume that `o` corresponds to 0 and `x` to 1.

VoteReprLeft{r}(v) This function takes only the left bit of each pair, so $\texttt{VoteReprLeft}\{101\}(3) = 1\,0\,1$. Likewise, $\texttt{VoteReprRight}\{101\}(3) = 1\,0\,0$. Clearly, `VoteReprLeft` and `VoteReprRight` correspond to what is left visible on the ballot that the voter takes with him.

Commit{k}(x) In cryptography, the commit primitive is equivalent to putting something in a sealed envelope, which can be opened, if so desired, or remain closed forever. A commit always needs a random input k. There are various ways to implement a commit: $\texttt{Commit}\{k\}(s) = \texttt{Encr}\{k\}(s)$ where `Encr` is some encryption function like AES, or $\texttt{Commit}\{k\}(s) = \texttt{Hash}(k; s)$ where `Hash` is a collision-resistent hash function. In practise SHA1 could be used.

DbleMix{p}(x) Here we assume the existence of only one Mix Authority. As is shown in [5], a more efficient verification procedure is possible if each authority actually runs two mixes. We call the first of these mixes A, and the second, B. We therefore get that $\texttt{DbleMix}(v) = \texttt{MixA}(\texttt{MixB}(v))$, where v is the vote.
Note that when using a mix, padding is necessary to avoid brute force attacks. Without padding an adversary could simply encrypt all votes and see whether the encryption matches some value. We will denote the padding for mix A as p_a and for mix B as p_b. When feeding something to the mix one also needs its public key, which we call a and b respectively, but which we do not include as parameters, since they can be considered constants (at least for the duration of an election).
We can therefore define $u = \texttt{MixB}\{p_b\}(v) = \texttt{RSA}_b(p_b; v)$ and $\texttt{DbleMix}\{p\}(v) = \texttt{MixA}\{p_a\}(u) = \texttt{RSA}_a(p_a; u)$ where p is the tuple $\langle p_a, p_b \rangle$.

Perm The permutation must be different for each ballot and so depends on the serial number. Of course, we assume the existence of some mechanism to create a random permutation of size t from a sufficient quantity of random bits. `InvPerm` is the inverse permutation of `Perm`.

4.3. The example ballot written more formally

The ballot of subsection 4.1 can be written as

$$\begin{aligned}
&\langle \texttt{Nr; VoteRepr}\{r_1\}(2); \texttt{Commit}\{k_{1L}\}(\texttt{VoteReprLeft}\{r_1\}(2); \texttt{DbleMix}\{p_{1L}\}(2)); \\
&\qquad\qquad \texttt{Commit}\{k_{1R}\}(\texttt{VoteReprRight}\{r_1\}(2); \texttt{DbleMix}\{p_{1R}\}(2)); \rangle \\
&\langle \texttt{Nr; VoteRepr}\{r_2\}(3); \texttt{Commit}\{k_{2L}\}(\texttt{VoteReprLeft}\{r_2\}(3); \texttt{DbleMix}\{p_{2L}\}(3)); \\
&\qquad\qquad \texttt{Commit}\{k_{2R}\}(\texttt{VoteReprRight}\{r_2\}(3); \texttt{DbleMix}\{p_{2R}\}(3)); \rangle \\
&\langle \texttt{Nr; VoteRepr}\{r_3\}(1); \texttt{Commit}\{k_{3L}\}(\texttt{VoteReprLeft}\{r_3\}(1); \texttt{DbleMix}\{p_{3L}\}(1)); \\
&\qquad\qquad \texttt{Commit}\{k_{3R}\}(\texttt{VoteReprRight}\{r_3\}(1); \texttt{DbleMix}\{p_{3R}\}(1)); \rangle
\end{aligned}$$

where `Nr`=795792618, `o`=0, `x`=1, $r_1 = 001$, $r_2 = 111$ and $r_3 = 110$. Beware that the meaning of r_i has changed: here the r_i refers to the value of r in the ith line (row) of the ballot, whereas in the previous section r_i was the ith bit of r.

We can describe an arbitrary row i of a ballot by using `InvPerm`. For instance, the value 2 in the first row corresponds to the image of 1 under the inverse permutation: $2 = \texttt{InvPerm}(1)$. In general, $v = \texttt{InvPerm}(i)$, where v is the voting option.

Combining this, an arbitrary row i reads:

$\langle$`Nr; VoteRepr`$\{r_3\}$(`InvPerm`$(i))$;
`Commit`$\{k_{3L}\}$(`VoteReprLeft`$\{r_3\}$(`InvPerm`$(i))$;`DbleMix`$\{p_{3L}\}$(`InvPerm`$(i))$);
`Commit`$\{k_{3R}\}$(`VoteReprRight`$\{r_3\}$(`InvPerm`$(i))$;`DbleMix`$\{p_{3R}\}$(`InvPerm`$(i))$);$\rangle$

From this it is rather trivial to see how the scheme can be generalized to T voter options, with T arbitrary.

5. The protocol

There are four parties involved in this protocol:

Voters Voters can verify that there vote is included.

Polling station The polling station are the authorities responsible for creation of the ballots, and communicating the side chosen by the voter to the tallying authority.

Tallying authority There must be at least one tallying authority, responsible for publishing the votes received on the internet for running a double mix. There can be more than one (double) mixing authority.

Auditor Both during the vote creation process as well as during the tallying process, some entity has to verify that the processes were conducted properly. Usually this role consists of creating a large collection of truly random bits and verify the consistency of the answers provided by the polling station, resp. the tallying authority.

Let's say that there are N voters, and that for each voter four ballots are created. Some time before the election, out of the set of $4N$ created and committed ballots, $2N$ ballots are chosen at random. The Polling Station must open the ballots chosen, showing all the random bits/keys/padding involved in creating them. Auditors and observers verify that there are no inconsistencies. The other $2N$ ballots are preserved for use on election day.

The true voting process is as follows:

1. When a voter votes, he chooses (or receives) a random, unused ballot.
2. He makes his choice i, i.e. selects an voting option (row) i with vote $v =$ `InvPerm`(i).
3. `Nr` and `VoteRepr` are printed and the voter verifies his vote(choice). If he made a mistake, the receipt must be invalidated or a new ballot must be used.
4. The voter chooses *Left* or *Right*. Consequently, the other side is made unreadable by overprinting. The voter leaves the booth with his (partially overprinted) receipt.
5. At the same time, the voting machine of the polling station opens `Commit`$\{k_{cS}\}$(`VoteReprSide`(v);`DbleMix`$\{p_{cS}\}(v))$, where $c =$ `Perm`(v) and S is the side, either L or R.
6. The contents of this commitment is published on the Internet; it is this first component `VoteReprSide`(v) that must correspond to the receipt obtained by the voter.
7. The second component, `DbleMix`$\{p_{cS}\}(v))$, is what goes into the mix.
8. The tallying authority applies the mix protocol and v is comes out.

9. The votes are tallied

Ballots that are not used at the end of the election can and should be completely opened.

Finally, to check the honesty of the mixes, random values of p_a and p_b are opened, but never p_a and p_b belonging to the same vote, as explained in [5].

Observe that the voting machine learns the voter's choice (obviously). Also, the Polling Station, who knows the value of `Perm` and the keys k_{cS} for the commits, can deduce the voter's choice. It must therefore be assumed that unopened parts are carefully destroyed.

Observe also that the authorities cannot cheat on the final result, except for an exponentially small probability of this going undetected and leading to an irrefutable proof when caught.

6. Conclusions

We have simplified Chaum's voting protocol, eliminating visual cryptography in favor of erasing some areas of the ballot by overprinting. We gave a formal description of this protocol.

From a theoretical point of view, schemes like Chaum's show that there are still many open questions and new avenues in voting protocols and voting technology. A potential problem of this scheme is that voters with a poor education may not be convinced that the receipt conveys no information and remain suspicious about the system. In fact, very naive people might become subject to manipulation if landowners (for instance) succeed in convincing them that the vote can be deduced from the receipt. In countries where large parts of the population are uneducated and illiterate, and where such manipulation is common, this will be an issue. And specifically with respect to Brazil: it remains to be seen whether protocols like this will be used during national elections, given the fact that a new law recently passed through the Brazilian Chamber and Senate abolished any kind of printing of the vote for auditing purposes.

Acknowledgements

David contributed enormously to this paper: by coming to Brazil twice and by giving lots of feedback on the issues presented here. Without his help this paper wouldn't exist. I also got positive feedback from Jeremy Bryans, Peter Ryan, Poorvi Vora, and the referees.

References

[1] Bryans, J., Ryan, P. *A Dependability Analysis of the Chaum Digital Voting Scheme.* CS-TR 809, July 2003. Available at `http://www.cs.ncl.ac.uk/research/pubs/trs/papers/809.pdf`.

[2] Chaum, D. *Secret-Ballot Receipts: True Voter-Verifiable Elections.* IEEE Security and Privacy, 2(1):38-47, 2004.

[3] Chaum, D. *Untraceable electronic mail, return addresses, and digital pseudonyms.* Communications of the ACM, 24(2):84-88, 1981.

[4] Vora, P. *David Chaum's Voter Verification using Encrypted Paper Receipts.* unpublished, `http://www.seas.gwu.edu/~poorvi/Chaum/chaum.pdf`, January 2004.

[5] Jakobsson, M., Juels, A., Rivest, R. *Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking.* USENIX Security '02, pp. 339-353. 2002.