# Remote ballot casting with Captchas

Stefan Popoveniuc, Poorvi L. Vora
The George Washington University
Department of Computer Science
{poste,poorvi}@gwu.edu

November 2, 2008

**Abstract**

A great threat to Internet voting is the possibility of an attacker writing a computer virus that spreads to enough computers in the world to elect by itself a winner regardless of the will of the voters. We present a technique that eliminates the possibility that computer programmers alone can meaningfully change votes cast using any computer, with or without malicious software running. The security of our technique depends on the hardness of Captchas – problems from artificial intelligence that are hard for computers but easy for humans. Our technique does not protect against attacks in which humans are involved.

## 1 Introduction

Ensuring that a computer is virus free is virtually impossible. The constant battle between the anti-virus industry and programmers that write malware is well known. Computer viruses can go undetected, because they can minimally impact the computer operation and can delete themselves immediately after they ran the first time. Operating systems, bootstrapping software, and other essential software can come with such a virus already installed and would avoid detection, since such a program has complete control over the computer, preventing any anti-virus software from inspecting it.

If a voting system uses computers and the Internet, casting a ballot on such a computer is a serious risk. The technique we propose allows any voter to confidently cast a ballot on any computer, without being afraid that a virus or any software can adversely influence the choices of the voter, without the help of a human. When voting in a polling place using stand alone Direct Recording Electronic (DRE) machines, the technique can eliminate the fear that the machines can learn the vote or change it to some specific candidate.

On a traditional DRE, it is desirable to let the computer know the exact votes, because the DRE needs to count the votes and print the tally at the end of the day. But in the case of other voting systems like Prêt à Voter [5]

or PunchScan [13], there is no need for the device to learn the clear vote, as the voter casts a coded vote and the tally is derived from all the coded votes, in a verifiable manner. Our technique is intended for such systems, generally being referred to as being *end-to-end verifiable*. They provide the voter with a receipt that contains an encrypted vote. All the voters can check that their votes were recorded properly (recorded as cast) by checking a public bulletin board. Everyone can check that the election authority correctly transformed the receipt on the public bulletin board into clear text ballots and tally them (tally as recorded).

DREs were introduced in the U.S. on a large scale after the Florida 2000 fiasco [18] because it is believed that voter intent is more clearly communicated using a computer. In paper-based voting systems, like traditional paper ballots, optical scan, or punch cards, the problem of determining the intent of the voter may be a source of endless debate. With computers, the voter gets feedback immediately after choosing a candidate, and correctly inferring the choice of the voters from the interaction is not a concern at all. Other reasons for using computers are related to their flexibility at the presentation layer: they can present a substantial number of ballot styles, so voters can go to any number of precincts; they accommodate a large number of languages; they can adequately accommodate voters with disabilities; they prevent over-votes and warn about undervotes.

One of the greatest problems with voting on a computer is privacy. Traditionally, the computer itself learns the clear text vote from the interaction with the voter. In turn, this may raise integrity concerns, because once the computer knows how a voter voted, it may decide on its own to flip the vote for some other candidate. If the computer used in the voter interaction would not learn the clear text vote, then it could change the vote at most in a meaningless or random way (e.g. giving a vote to a random candidate).

A big obstacle in remote electronic voting (Internet voting) is the ability of a virus to capture the voter choices and harm the computer if the choices are considered "bad" (reboot before casting, delete the hard drive, etc.). This is possible because the virus has complete control over the device and learns the votes while the voter makes her selection. The virus could also change the vote and still present the voter with the "Your vote has been correctly cast" screen, giving the voter a false sense of comfort. This concern has been repetitively raised in reports [7][9] and is considered one of the biggest impediments in the adoption of Internet voting (along with denial of service attacks and improper influence). Improper influence refers to a voter voting in the presence of another person, a coercer that exercises pressure in order for the voter to cast a ballot in a certain way. A denial of service is an attempt to make a computer resource unavailable for its rightful users.

Starting to think about how to solve the malicious software problem may take us one step closer to Internet voting by eliminating or mitigating a great obstacle to its adoption. We suggest associating instances of a Captcha [16] with the candidates on a ballot. To cast a ballot for her favorite candidate the voter has to solve the Captcha associated with that candidate. We suggest using this

method in the context of an end-to-end, software independent voting system.

The problem we are approaching is viewed as being very difficult. The SERVE report [7] states that "There really is no good way to build [a secure, all-electronic remote voting system] without a radical change in overall architecture of the Internet and the PC, or some unforeseen security breakthrough". We suggest a technique that takes us one step closer to a solution for such a problem. The technique requires the use of Captchas that are secure enough, and requires the existence of a large number of instances of Captchas. This may be viewed as the cost of the solution to a problem that is widely recognized as being very difficult.

## 2 Security Model

We briefly state the assumptions that our technique is based on. While our assumptions may not hold for any Captcha that is currently in use, it may be that there will exist Captchas that satisfy our assumptions.

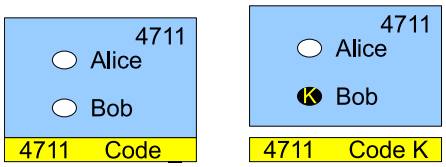
First we assume that a Completely Automated Public Turing Test to Tell Computers and Humans Apart (Captcha), exists . It is a program that can generate and grade tests that most humans can pass but current computers cannot. In particular, its security can be based on the hardness of a problem in artificial intelligence for current computers, and depends on the problem being easy for humans. A simple example is the classical Captcha [16] that is present on popular sites like Yahoo or Google to prevent, for example, automatic creation of accounts.

Secondly, we assume that, given a set of Captchas and a solution for one of them, no program can determine with a non-negligible advantage, to which of the Captchas the solution belongs to, or does not belong to.

Third, we assume that any human can solve a Captcha most of the time. When a voter believes that she cannot solve a Captcha, or the computer that generated it noticed that the solution provided is not valid, the voter gets a new ballot with a new set of Captchas on it.

Fourth, we assume that the facility that produces the ballots does not collude (i.e. share more information than strictly needed) with the device used for showing the ballot to the voter. In other words, the computer that is connected to the Internet and used by the voter to vote, receives only the ballot from the server, and nothing else. If the election authority that produces the electronic ballots is colluding with the malware on the voter's computer, both integrity and privacy can be compromised. Also, we assume that the order of the candidates on the ballot is fixed and publicly known.

Our technique does not protect against attacks that involve Captcha farms. A virus can captures the ballot and re-transmit it immediately to people hired to decrypt Captchas and to cast ballots for a certain candidate. In such scenario a fairly large number of people should be engaged in this attack, each Captcha must be solved by the human attacker before the connection session expires and the attacker has to be faster than the voter. Dan Wallach [17] states that an




(a) A sample Scantegrity ballot, just like a regular optical scan ballot.

(b) A voted Scantegrity ballot. When the oval is filled in, a confirmation number appears. The receipt may contain the serial number and the confirmation number

Figure 1: Scantegrity ballot

attack on the integrity of an election ran via the Internet would take O(1), a single individual; even if Captcha farms are used, our technique improves this estimation to O(P), the number of individuals required to solve the Captchas.

# 3 Description of the technique

In this paper we describe a technique that can be used in association with the back-end of Scantegrity, an end-to-end verifiable voting system based on optical scan ballots. Our paper focuses on the modification that enables a system such as Scantegrity to be used for remote voting purposes; thus it focuses on the casting of the ballot. For a brief description on the mechanism that allows end-to-end verifiability see section 4.

In Scantegrity II [3], a precinct-based, optical scan cryptographic end-to-end auditing system, each candidate has a confirmation code that is revealed when a vote for that candidate is cast (see Figures 1(a) and 1(b)). All the confirmation codes are pre-committed to, and the voted ones are run through a mixnet-like mechanism that transforms them into clear text votes in a publicly-verifiable manner. We start by describing the technique in the context of Scantegrity II. For our work, it is irrelevant how ballots are counted, as our technique addresses only the casting phase.

In this paper, each possible vote is associated with a Captcha; Captchas are not repeated within one ballot. A voter votes by solving the Captcha corresponding to the candidate of her choice. If the assumptions in section 2 hold, a virus cannot cast a vote for a particular candidate, because it cannot solve the Capthca associated with it.

The reader should note that Captchas based on the ability of a human to read a distorted text that a computer cannot read is just a particular example of a Completely Automated Public Turing Test to Tell Computers and Humans Apart. Other ideas can stand behind constructing such tests, e.g. recognizing

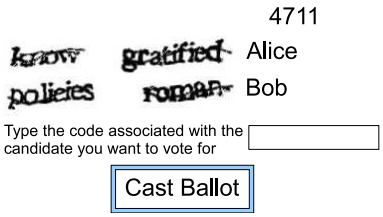



Figure 2: Scantegrity ballot with Captchas. Captchas compliments of the re-Captcha project - www.captcha.net

an object or a being in an image, creating a description of a picture, saying if a piece of text makes sense or not, etc. We hope that the future will bring more practical examples of such tests that humans can easily solve, but computers cannot.

### 3.1 The voting ceremony

Using any computer connected to the Internet, the voter accesses the website of the election authority and authenticates herself (the authentication mechanism is out of the scope of this paper). The voter is then given a ballot that for each race has a set of Captchas associated with every candidate (see Figure 2 for an example of a text based Captcha). For each race, the voter provides the solution to the Captcha (e.g. types in the text that she can read next to her favorite candidate), and presses the cast ballot button. If the voter is visually impaired, she can use an audio Captcha. Thus the technique uses a *cognitive* channel between the voter and the server to provide the voter with the codes corresponding to the individual candidates. If certain AI problems are hard for the computer, this channel cannot be tapped by the computer and hence it cannot (a) determine with certainty what the vote is, and (b) change the vote in a meaningfully. After casting her ballot, the voter receives a signed privacy preserving receipt from the server, that contains the text that the voter typed in. She keeps the receipt to later check that her vote is correctly posted on the bulletin board.

To determine what the vote is, a virus would have to associate the solution of the Captcha provided by the voter (e.g.the string entered) by the vote with a candidate. If this problem is hard, and if a program cannot do so, the vote is private. Further, to vote for another candidate, a virus would have to correctly guess the solution of the hard AI problem corresponding to the candidate; that is, it would have to break the Captcha, or get the solution by other means.

If the server happens to receive a string that is not in any of the Captchas for that race, it will trigger an alarm and will communicate to the voter that something has gone wrong. The channel used for this alarm can be computer independent (voice, sms, mail etc). Thus, if the Captcha cannot be solve by the virus in the small amount of time that the server is willing to accept a vote using

the ballot, no software on the computer can affect the integrity of the election.

If the voter can choose more than one candidate, as in approval or plurality voting , she can simply provide the solution for all the Captchas associated with her favorite candidates. If rank voting is used, the voter can associate the solution to a Captcha to her first choice, another solution to her second choice, and so on. For range voting each score that can be assigned to a candidate has to have a Captcha associated with it. Our technique does not apply to write-ins.

For text bases Captchas, using an alphabet with $n$ symbols and words of length $k$ there are $n^k$ unique confirmation codes and thus possible Captchas[1]. Assuming a race has $x$ candidates, the probability that a virus correctly guesses the text used in any Captchas on a given ballot is $P = \frac{x}{n^k - 1}$ (the voter already solved a Catpcha). A practical example is to use words of length eight ($k = 8$) consisting of symbols from an alphabet with thirty two symbols in total ($n = 32$, 26 letters and 6 numerals). For a contest with ten candidates ($x = 10$) the probability that anyone guesses a text that appears on a particular ballot ($P$) is less than one in one hundred millon (assuming a random guess). Thus, it is very unlikely that a virus can simply guess a valid code. Note that this probability would be larger if it were required that the Captchas represent valid words in the language, as in our example

If the malware on the voter's computers has some knowledge of how the voter is going to vote, it can rearrange the Captchas on the ballot (without knowing what they are). For example, the virus has some knowledge that the voter is going to vote for Alice, because the voter visited Alice's web site, donated money online, or because it analyzed the content of her inbox. The virus wants to trick the voter into casting a vote for Bob, so it modifies the ballot received from the server by taking the Captcha associated with Bob and putting it next to Alice.

To avoid such an attack, a meta Captcha can be constructed: a Captcha that contains Capthcas such that no software can rearrange the individual Captchas. For example, all Captchas in a single race are placed on common meaningful background (e.g a picture of a tree). For text based Captchas, the malware cannot clip only the text from the Captcha and paste it somewhere else in the background picture, because it does not know the exact boundaries of the text in the Captcha. Trying to clip a regular area (such as a rectangle) is going to perturb the background and the voter will notice it. Similarly, in audio Captchas, a musical background musical could be used; if pieces are clipped out an rearranged, the background music would sound different. Like in any other voting system where the order of the candidates is fixes, we assume that the order of the candidates is publicly known (perhaps alphabetical). We assume voters would notice if the order of the candidates is different from the official posted one.

A related concern is a forced randomization attack. A coercer can instruct a voter to bring a receipt that is formed in a certain way (e.g. starts with letter A). Like other voter verifiable schemes (e.g. PunchScan, Prêt à Voter), our

[1] a good Captcha would have all the possibilities equally hard to break and indistinguishable for any software

proposal does not protect against such randomization attacks.

Another attack that we avoid relates to the server that received the coded votes seeing how the voter voted. The server constructs the Captchas on the ballot and sends the ballot to the voter. The server then receives the result of the test that corresponds to the chosen candidate (e.g. the text in the Captcha). At this moment, the server knows how the voter voted, and interrupts the protocol, or gives the voter a badly formatted receipt. The voter does not get any receipt or gets an invalid one. To stop the server from rejecting votes selectively, we use blind signatures [2]. When the voter presses the "Cast Ballot" button, a piece of software (provided by a third party) blinds all the selections and sends them to the server. The server signs the votes, not knowing what they are, and returns them to the client, who now un-blinds them, obtaining a signed receipt of the selection which is sent to the server, in the final step of the protocol. The server cannot reject or change the votes because the voter provides a signed receipt of the vote. Note that the voter has now a receipt with all her selections, signed by the election authority. If she later checks the official website and notices that her selections are not recorded properly, she has her signed receipt, which is an irrefutable proof for the malfeasance. Unlike other usage of blind signatures, ours does not require an anonymous channel; all the communication takes place in a single session and is authenticated. This is because, in our scheme, the vote is encoded.

The voter needs to be sure that the ballot she receives is properly constructed by the server (in [3] this is referred to as the printing audit). This is done using a cut and choose protocol. The voter may choose to spoil the received ballot and the server would then have to publicly reveal all the information with regards to that ballot. There are two ways this can be implemented. First, the server can simply sign all the ballots that it hands out. If it turns out that the ballot is not constructed properly, the voter has irrefutable proof (the signed blank ballot) of the malfeasance. Another way of solving this problem is that the server hands out to the voter an unsigned ballot, the voter decides to spoil the ballot and send it to the server using a blind signature protocol. The server has to sign the ballot because it does not know if it signs a vote or a spoiled ballot. The server may authenticate itself to the voter via a technique proposed in [15], which is an alternative to digital signatures.

When digital signatures are used on a potential infected computer, the malware can interfere with the verification of the signature. In the context of our solution, this would disrupt the voting process, equivalent to a denial of service attack, but with the focus on the clients rather than the traditional denial of service attacked directed to the server. We assume the only scope of the virus is to either find out how to voter voted or to change the vote. Our solution does not address denial of service attacks, which can be mounted in a variety of ways without infecting hosts.

As mentioned in subsection 3.4, the proposed technique does not address the possibility of a human coercer, neither next to the voter nor remote, but only mass coercion by a virus that can go undetected while infecting most of the voting hosts. It also does not address the instance when humans are used

to solve Captchas and to change the vote.

## 3.2 Accessibility

Visually impaired voters can use an audio version of a Captcha. A test is delivered to them via voice, and they have to solve the test to prove that they are humans and not software. Many popular web sites (www.gmail.com, www.hotmail.com) use audio Captchas in addition to visual ones, for accessibility reasons. Some attacks have been targeted specifically at audio Captchas [19], with a higher rate of success than with their visual equivalent. It is possible that this is due to the fact that visual Captchas have received considerably more attention than have audio Captchas, and that more research in the design of audio Captchas will result in more secure audio Captchas. Further, audio and visual Captchas can be designed around hard problems in language processing, with the same hard AI problem – summarizing whether a given sentence is about "rain" or about "snow" for example – represented either visually or in audio for a voter, based on the voter's preference.

### 3.2.1 Illiterate voters

We describe a simple Captcha based technique, inspired by PunchScan, that can be used by illiterate voters. We assume that the order of the candidates is fixed and that the voter can recognize her favorite candidate. The voter can memorize the way the candidate is written, or the candidate can have a graphical representation, e.g. a symbol or a color. With each candidate is associated a symbol that cannot be interpreted by a computer program, but can easily be interpreted by a human voter, e.g. a picture of a dog. At the bottom of each contest there is a big picture containing symbols that represent the same concepts as the symbols associated with the candidates. To vote for a particular candidate, the voter has to select the symbol on the picture that represents the same concept that is represented by the symbol next to her favorite candidate. The symbols, their association with candidates and the locus in which they appear on the bottom picture are all uniformly random.

We give a simple concrete example of such a ballot: there are two candidate on a ballot, Alice and Bob. Alice has a picture of a black cat associated with her and Bob of a brown dog. At the bottom of the ballot there is a picture containing a white cat and a red dog. To cast a vote for Alice, the voter would have to click on the white cat. The assumption is that a computer program cannot distinguish a picture of a cat from the picture of a dog, nor can it detect where the symbols are placed on the bottom picture.

## 3.3 Captcha Security and the privacy and integrity of the voting system

Unlike some hard problems in computer science, scientists agree that AI problems will eventually be solved. However, we do not require the Captchas to be

secure forever for the system to be secure against attacks on the integrity of the vote tally. If the Captcha cannot be broken in the short amount of time of a voting session, then the device cannot figure out how the voter voted, and cannot meaningfully change the vote. The receipt that the voter gets is a record that the vote was properly recorded and thus is a proof of integrity.

On the other hand, if Captchas are broken in a longer time frame, then a computer virus can figure out how the voter voted, breaching privacy, but not directly affecting the integrity of the election.

Recently, there appeared some attacks [21], [20] that solve with high probability some forms of visual Captchas based on the user recognizing the distorted text on an image. Other attacks have been aimed at audio Captchas [19]. At the same time, new forms of tests that are claimed to be solvable only by humans appeared [1]. We urge the reader to treat the specific Captchas used by us only as examples of the manner in which Captchas will be used. We expect that any implementation of our system will use the most secure Captchas available at the time an election is ran.

### 3.4 Potential attacks

The purpose of our work is to stop a team of computer programmers from creating a virus that would manipulate an election. We achieve this by preventing the voting machine from learning the vote while the voter is using it (and possibly from ever learning the vote). Our technique does not guard against a human looking at the votes. The human can sit behind the voter, or can remotely receive the voted ballot from the voting device. This kind of attacks may have scaling problems, since a large group of people needs to be involved to coerce or modify (in real time) a significant fraction of all votes.

The voter can take a screen shot of the ballot and bring it to a coercer. If the coercer is a human, our technique does not protect against such attacks. Two observations are worth making: first, the screen shot can be altered by the voter to produce a convincing proof to the coercer, and second, this attack is valid for any voting system, be it remote (as in mail-in voting) or precinct-based (taking a movie of yourself voting).

It is worth mentioning that a man in the middle attack does not make any sense, if run by a computer program and not a human - a machine in the middle attack. The machine in the middle gets the ballot from the server and gives the voter another ballot (produced by the attacker). The machine in the middle would then learn how the voter intends to vote, but cannot do anything about it. It cannot change the vote, nor can it transmit it unaltered to the server (because it cannot read the Captcha on the real ballot). Thus this attack would not stop the voter from voting, because the server would allow another ballot to be cast later on, since the first session was unsuccessful from the point of view of the server.

# 4 Producing and checking the tally

We briefly describe a known technique originally presented in [4] that allows each voter to check that her vote was correctly included in the final tally.

Let $N$ be the number of ballots in an election and let $c$ be the number of candidates on a ballot. Consider three tables: $R$ (stands for receipt values) contains coded votes; $T$ (stands for tallies and results) contains clear text votes that are countable by anyone; $D$ (stands for decrypt) connects $R$ with $T$. $R$ is a matrix with $N$ rows and $c$ columns, each row represents a ballot. $R$ is a matrix with $c$ rows and $N$ columns, each row represents a candidate. An element $(i, j)$ is either marked or not marked in $R$ and $T$, a mark corresponds to a vote for the candidate. $D$ is a blob with $N \times c$ elements (the number of rows and columns is irrelevant). Figure 3 gives an example of the three tables for an election with six ballots and two candidates.

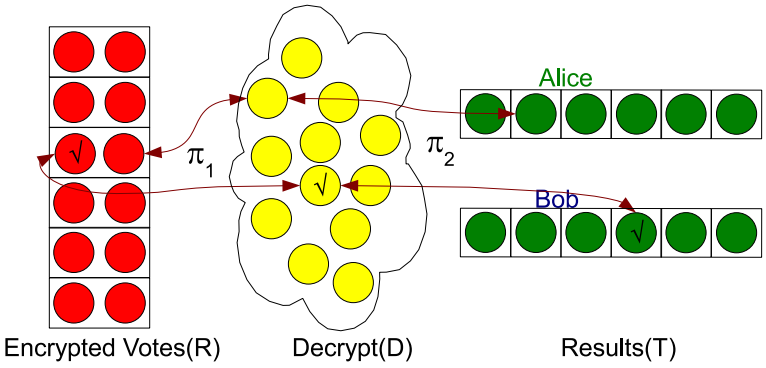



Figure 3: Pointer-based mixnet

The tables are connected by two permutations, $\pi_1$ and $\pi_2$. $\pi_1$ connects $R$ (receipts; coded votes) with $D$ (decrypt): $D_k = R_{\pi_1(k)}$, where $k$ is some canonical representation of $(i, j)$; for example, $k = (c-1)i + j$. $\pi_2$ similarly connects $D$ with $T$ (results or tally): $T_k = D_{\pi_2(k)}$. These permutations are constrained to return a mark for a particular candidate in $R$ to a mark for the same candidate in $T$.

For two candidates, the properties of the permutation may be formalized as follows: let $\pi_1 : [0, 1, ..., N \times c - 1] \rightarrow [0, 1, ..., N \times c - 1]$ be bijective and let $\pi_2 : [0, 1, ..., N \times c - 1] \rightarrow [0, 1, ..., N \times c - 1]$ be bijective such that no two elements belonging to the same ballot initially (in the same row in the initial set) are mapped to two elements belonging to the same candidate (the same row in the final set):

$$\forall i, j, i \neq j \; having \; [i/c] = [j/c] \Rightarrow [\pi_2(\pi_1(i))/b] \neq [\pi_2(\pi_1(j))/b] \qquad (1)$$

where $[x]$ represents the greatest integer less than or equal to $x$. Note that no group operation (such as modulo addition or permutation composition) is

performed on the payload. For $c > 2$, the condition requires that the remainder on division by $c$ (the candidate for a mark or nomark) be preserved; because the rows and columns are reversed, one may then wish to view table $T$ canonically as listed column by column, and $R$ row by row.

Initially, the election authority publishes commitments to $\pi_1$ and $\pi_2$, files in the $R$ table with solutions to Captchas and publishes commitnets to each one. To check the correctness of this step, an auditor can request some statistical significant number of ballots to have their commitments opened (both the Captcha solutions and the corresponding positions in $\pi_1$ and $\pi_2$ all the way to the $T$ table). When a cast ballot is received, the election authority opens the commitment associated with it in the $R$ table. Any voter can now check that her ballot is correctly posted. After the polls close, the final audit checks that one of the two properties hold: $D_i = R_{\pi_1(i)}$ or $D_i = T_{\pi_1^{-1}(i)}$ and that the properties of the two permutations $\pi_1$ and $\pi_2$ hold; more precisely it is statistically checked that both $\pi_1$ and $\pi_2$ are injective functions and that Equation 1 holds. Because the voting system cannot predict which property will be checked, a successful audit implies that both properties hold with high probability. The privacy is preserved, since no complete link is revealed from the $R$ table to the $T$ table.

# 5 Previous work

Shamos [14] mentions the necessity of having a large piece of Optical Character Recognition (OCR) software to recover the votes from a ballot that is constructed from images, but does not go the full way to propose the hardness of the OCR process.

Our work is closely related to Hanley et al. [6], and Oppliger et al. [11]. Hanley's technique is the first to suggest Captchas in the context of voting. The focus of their work is on defending against network attack by creating a single Captcha with all the names of the candidates written in wavy text. Our technique is proposed in the context of end-to-end voting systems, in which the voters get a receipt and can ensure that their vote was recorded properly, and everyone can check that the tally is correctly constructed from the recorded receipts. Instead of inventing an entire new voting system, we only suggest changing the front-end (ballot presentation layer) on a voting system that is known for having provable integrity properties, as opposed to trusting the server for the integrity of the election. We suggest using a blind signature protocol to protect against the server selectively rejecting ballots. We make the important remark that using Captchas protects from potential virus attacks on the voters computer, in addition to attacks on the communication network itself. Our technique allows for having the names of the candidates in a standard manner in terms of order, font and size, which may an important legal aspect.

Using Captchas for voting has also been suggested [15], but for a different purpose: to allow the voter to safely hand her receipt to a trusted organization that is going to perform the checks for the voter.

"The Voting Ducks" [10] and Joaquim et al. [8] use the concept of a second

channel, independent of the computer, that is used to transmit to the voter the codes for the candidates (unique per ballot) and the voter would type in the code on the computer. Paul et al. [12] describe an authentication method using visual cryptography that can be used for casting a ballot on a potential infected computer, by having the ballot constructed as a large image, with the candidates spread across the image. While these techniques would prevent the computer from ever seeing the vote, they all use a channel that is independent of the computer (e.g. sms or postal mail) and thus the practicality is debatable.

# 6 Conclusion

In the context of end-to-end voting systems, we have described a simple technique for stopping a computer program from learning how a voter votes or changing the voter's choices. The technique suggests associating Captchas to the candidates on the ballot. The voters have to solve the Captcha associated with their favorite candidate. This prevents the device from figuring out the votes or from changing them. The technique is equally effective for safeguarding against an eavesdropper that listens to the transmission medium passively, or a machine in the middle attack. Our solution addresses a major obstacle to the adoption of Internet voting.

# References


[1] alipr.com. Next-generation captcha exploits the semantic gap. `http://tech.slashdot.org/article.pl?sid=08/04/23/0044223` [Online; accessed 2-June-2008].

[2] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto'82*, 1982.

[3] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman. Scantegrity ii: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology on USENIX/Accurate Electronic Voting Technology Workshop*. USENIX Association, 2008.

[4] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. T. Sherman, and P. Vora. Scantegrity: End-to-end voter verifiable optical-scan voting. *IEEE Security and Privacy*, May/June 2008.

[5] D. Chaum, P. Y. A. Ryan, and S. Schneider. A practical voter-verifiable election scheme. In *In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, ESORICS, volume 3679 of Lecture Notes in Computer Science*, pages 118–139. Springer, 2005.

[6] D. Hanley, J. King, and A. dos Santos. Defeating malicious terminals in an electronic voting system. In *Proceedings of the SBSeg*, September 2005.

[7] D. Jefferson, A. D. Rubin, B. Simons, and D. Wagner. A security analysis of the secure electronic registration and voting experiment (SERVE). `http://www.servesecurityreport.org/`, January 21 2004. Technical Report.

[8] R. Joaquim and C. Ribeiro. Codevoting: protecting against malicious vote manipulation at the voter's PC, 2007. `http://kathrin.dagstuhl.de/files/Submissions/07/07311/07311.JoaquimRui2.ExtAbstract!.pdf` [Online; accessed 19-October-2007].

[9] T. Kohno, A. Stubblefield, A. D. Rubin, and D. S. Wallach. Analysis of an electronic voting machine. Technical Report TR-2003-19, Johns Hopkins Information Security Institute, July 23 2003.

[10] W. U. of Technology. The voting ducks voting system, 2007. `http://www.vocomp.org/teams.php` [Online; accessed 19-October-2007].

[11] R. Oppliger, J. Schwenk, and C. Lhr. Captcha-based code voting. In *3rd International Conference on Electronic Voting*, EVOTE08.

[12] N. Paul, D. Evans, and A. Rubin. Authentication for remote voting. In *Workshop on Human-Computer Interaction and Security Systems*, Fort Lauderdale, Florida, April 2003. `http://www.cs.rice.edu/~dwallach/pub/remote-voting2003.pdf`.

[13] S. Popoveniuc and B. Hosp. An introduction to PunchScan. In *IAVoSS Workshop On Trustworthy Elections (WOTE 2006)*, Robinson College, Cambridge UK, June 2006.

[14] M. I. Shamos. Paper v. electronic voting record - an assessment. In *Proceedings of the 14th ACM Conference on Computers, Freedom and Privacy*, April 2004. `http://euro.ecom.cmu.edu/people/faculty/mshamos/paper.htm`.

[15] R. Simha and P. L. Vora. Vote verification using captcha like primitives. In *IAVoSS Workshop On Trustworthy Elections (WOTE 2007)*, University of Ottawa, Ottawa, Canada, June 2007.

[16] L. von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard AI problems for security. In *Proceedings of Eurocrypt*, pages 294–311, 2003.

[17] D. Wallach. Voting system risk assessment via computational complexity analysis. *William and Mary Bill of Rights Journal*, December 2008.

[18] Wikipedia. United States presidential election in Florida, 2000 — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/United_States_presidential_election_in_Florida_2000` [Online; accessed 02-April-2008].

[19] WintercoreLabs. Breaking gmails audio captcha. http://blog.wintercore.com/?p=11 [Online; accessed 2-June-2008].

[20] J. Yan and A. S. E. Ahmad. Is cheap labour behind the scene? - low-cost automated attacks on yahoo captchas. http://homepages.cs.ncl.ac.uk/jeff.yan/yahoo.htm [Online; accessed 2-June-2008].

[21] J. Yan and A. S. E. Ahmad. A low-cost attack on a microsoft captcha. http://homepages.cs.ncl.ac.uk/jeff.yan/msn_draft.pdf [Online; accessed 2-June-2008].