Making the World Wide Web Safe for Democracy: Using Practice Ballots to Prevent Vote Buying in Internet Elections

Adam Wolf
Advisor: Professor Michael Fischer
May 3, 2004

Abstract

One of the major problems with internet voting is the ease with which votes can be bought and sold. In a traditional election, the secret ballot affords some protection against vote buying. A would-be vote buyer can pay a voter to choose a certain candidate, but the buyer has no way of verifying that the voter chose who he was paid to choose. In other words, the buyer cannot know if he got what he paid for. Unfortunately, if we move to the arena of internet voting, this doubt disappears. A voter could anonymously sell his credentials to a vote buyer, or proxy with a vote buyer to demonstrate that his ballots were cast for a certain candidate. There is only a guarantee of secrecy if the voter chooses to exercise this right.

In order to reintroduce the doubt that was present in the traditional election, I propose the creation of “practice” ballots. These ballots would be indistinguishable to all except the government - to voters and vote buyers alike - and any voter can obtain an unlimited number of practice ballots. The practice ballots would be accepted exactly as normal ballots, but the votes would not be counted in the final tally. Because the two types of ballots are indistinguishable, we can reintroduce doubt. A voter could sell his practice ballot to a vote buyer and then use his real credentials to vote for whomever he wanted. Again, the vote buyer does not know if he got what he paid for. I propose a protocol relying on the properties of quadratic residues and RSA blind signatures, which allows the government to distinguish real and practice ballots, but prevents it from being able to link ballots to voters, thus maintaining privacy.
1 Introduction

“Vote selling is a problem in all elections, but it is a special concern for Internet voting, since the Internet can facilitate large scale vote buying and selling by allowing vote buyers to automate the process” [1]. If Internet voting is to become a viable option, we must have a way of preventing vote buying.

In a traditional election, a voter casts her ballot alone in a private booth. Thus, while someone may attempt to buy a vote, he has no way of knowing if he got the vote he paid for. On the internet however, there is no guarantee that the voter performs the voting by herself in secret; she could easily use a proxy to allow someone else to watch her vote. There is not even a guarantee that the voter is the one casting the ballot; she could simply sell her credentials themselves. What needs to be done is to reintroduce the doubt of the buyer that he is getting what he paid for.

In order to do this, I introduce the idea of “practice” credentials. Any voter may request from the government as many practice credentials as she wishes. These practice credentials would allow a user to login and cast a ballot as he would were he using real credentials but the votes cast using the practice credentials would not count. The initial obvious benefit is that voters unfamiliar with the internet voting system would have a chance to try the system out before using it for votes which would actually count, or even to have someone show them how to use the system all the way through without that helper being able to see their actual votes. Furthermore, political parties or independent election monitors could check to see that all candidates are being listed on the ballot. But most importantly, practice credentials will look identical to real credentials to any user, including a prospective vote buyer; the two types would be distinguishable only to the official government server. Thus, an internet voter could sell practice credentials to a would-be vote buyer without any benefit to the buyer. Or, a voter could proxy with the vote buyer while using practice credentials, and the exchange would look identical to an exchange using real credentials.

With the use of practice credentials, the element of doubt which had been a deterrent to vote buying in the traditional system is reintroduced to internet voting. Furthermore, a user could sell as many of these credentials as she likes to the vote buyer, whereas in the traditional system, a voter could only sell her vote once. As a result, the deterrent is even stronger, as a vote buyer could end up spending all of his budget on one clever voter, and still gain no benefit.

In Section 3, I discuss a protocol which will allow such practice ballots to exist, while maintaining the voter’s privacy. In Section 4, I discuss a reference implementation using this protocol. Section 5 compares the security properties of this reference implementation to those of SERVE, an internet voting system, which until recently, was to be used for the 2004 general elections by the Department of Defense.
2 Background

Before going into the details of this system, I will discuss some of the cryptographic primitives necessary for such a system.

2.1 RSA Signatures

2.1.1 Traditional RSA Signatures

An RSA signature relies on two functions, $S_d(m)$, a signing function which requires knowledge of a private signing key $d$, and $V_e(m, s)$, a verification function which requires knowledge of a public verification key $e$. For RSA, the functions are as follows:

$$S_d(m) = m^d \mod n$$
$$V_e(m, s) = (s^e \mod n) = m$$

$n$ is called the modulus and is public and the product of two large primes $p$ and $q$. $e$, the signing key, and $d$, the verification key are related by the property $ed \equiv 1 \pmod{(p-1)(q-1)}$. Group theory tells us that given these properties of $e$ and $d$, $x^{ed} \equiv x \pmod{n}$. So, $V_e(m, S_d(m)) = (m^d) = m$, which returns true. We also believe, and experience has taught us, that given $e$ and $n$, it is hard to find $d$ without knowing how to factor $n$ into $p$ and $q$. Thus, only those who are given the secret $d$ can validly sign messages; the signatures cannot be forged.

2.1.2 Fully-Blind RSA Signatures

A fully blind signature is a signature in which the signer cannot read the message he is signing. This is tantamount to asking someone to sign on the outside of an envelope, without having read the letter inside. We can create such signatures using RSA using a system originally developed by Chaum. In this statement of the protocol, Alice has a message she wants signed by Bob, but she doesn’t want Bob to read the message. Bob has RSA signing key $d$ and modulus $n$ and Alice has verification key $e$, modulus $n$, and message $m$. The steps are as follows:

1. Alice generates a random number $r$, called a blinding factor. She computes $z = r^e m$ and sends $z$ to Bob.

2. Bob signs $z$, computing $S(z) = z^d$ and sends $S(z)$ to Alice.

3. Alice “unblinds” the signature, by computing $r^{-1}S(z) \mod n = r^{-1}z^d \mod n = r^{-1}r^ed = m^d = S(m)$.
Thus, Alice now has a valid signature on $m$. Bob cannot read $m$ without knowing $r$; to him, he just signed a random string [2].

### 2.1.3 Partially-Blind RSA Signatures

The problem with the above system is that Bob has no idea what he is signing. We would like a system in which Bob at least knows the form of what he is signing. We can create a system in which Bob can be sure to a high probability of the form of the message he is signing. It works as follows:

1. Alice generates $k$ messages, $m_1, ..., m_k$ and $k$ blinding factors, $r_1, ..., r_k$. Alice computes the corresponding $z$s, $z_i = r_i^e m_i$ and sends these to Bob.

2. Bob chooses $k - 1$ of the messages at random to challenge.

3. Alice sends in $r_i$ and $m_i$ for each message Bob challenged.

4. Bob verifies that the $m_i$ are of the form he requested, and that $z_i = r_i^e m_i$ for each $i$.

5. If Bob verifies this properly, he signs the unchallenged message, sending Alice $S(z_j)$.

Only if Alice guesses correctly which message Bob will challenge, can she get a signature on a message of invalid form. The probability of guessing correctly is $1/k$ [2].

### 2.2 Quadratic Residues

A quadratic residue mod $n$ is a number $y \in \mathbb{Z}_n^*$ s.t. $y = x^2 \mod n$ for some $x \in \mathbb{Z}_n^*$. In other words, quadratic residues are elements which have square roots. We call those that do not have square roots quadratic non-residues. We use the Legendre symbol $(\frac{x}{p})$ to test if a number $x$ is a quadratic residue mod $p$, where $p$ is a prime. $(\frac{x}{p}) = 1$ if $x$ is a quadratic residue mod $p$, and $(\frac{x}{p}) = -1$ if $x$ is a quadratic non-residue mod $p$. For $p$ a prime, exactly half of the elements of $\mathbb{Z}_p^*$ will be quadratic residues and exactly half will be quadratic non-residues. If we let $n = pq$, the product of two primes, we know by the Chinese Remainder Theorem that we can think of $\mathbb{Z}_n^*$ as $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$. Thus, we have 4 classes of numbers. We will call numbers which are quadratic residues mod $p$ and $q$ $Q^{11}$, numbers which are quadratic non-residues mod $p$ and $q$ $Q^{00}$, numbers which are quadratic residues mod $p$ but not mod $q$ $Q^{10}$, and numbers which are quadratic residues mod $q$ but not mod $p$ $Q^{01}$. It turns out that
these 4 classes also come in equal quantities. It also turns out, that without knowing the factorization of \( n \), \( Q^{11} \) and \( Q^{00} \) are indistinguishable. This will turn out to be an important property later on [9].

### 3 Preventing Vote Buying and Maintaining Privacy

While the existence of practice ballots does serve as a deterrent to vote buying, it is also important to keep a voter’s ballot private. The government should not be able to determine who voted for whom. In a traditional election, the sign-in information is kept separate from the ballots, and in an absentee election, the identifying information is separated when the outer envelope is removed. We need an equivalent separation for an internet election. For a standard internet voting system, Peralta proposed a protocol which would allow ballots to be disassociated from credentials. I propose an adaptation of that scheme for our purposes, which allows practice and real ballots to be distinguished without sacrificing the voter’s privacy.

#### 3.1 Peralta’s Scheme

First, I will describe Peralta’s scheme[8]. The steps are as follows:

1. Voter sends in the combination of her ballot and a random nonce blinded for RSA signature along with her credentials.
2. Government signs the ballot/nonce combination, and sends it back to the voter.
3. Voter unblinds the ballot/nonce combination.
4. Voter sends this combination in anonymously.

Using this protocol, the government can check the voter’s credentials, signing only the ballots for voters who are eligible to vote and have not yet voted in this election. However, the government cannot see the ballot at this time. When the ballot is submitted, the government can record the nonce, which prevents multiple submissions of the same ballot, without tying the nonce to the voter, as the nonce was blinded at the time of signing.
Here are the precise mathematics of the system:

<table>
<thead>
<tr>
<th>Voter</th>
<th>Government</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Choose random $n, r \in \mathbb{Z}_n^*$&lt;br&gt;L et $m = b \oplus n$&lt;br&gt;Compute $z = r^{e_m}$, credentials $\mapsto$ (mod $n$)</td>
<td>Authenticate credentials&lt;br&gt;Issue $S(z) = z^d$ (mod $n$)&lt;br&gt;$S(z) \leftarrow (r^{e_m})^d = r^{ed} m^d = rm^d$</td>
</tr>
<tr>
<td>2. Compute $S(m) = m^d = r^{-1}S(z)$ (mod $n$)</td>
<td></td>
</tr>
<tr>
<td>3. Submit anonymously $S(m)$</td>
<td></td>
</tr>
</tbody>
</table>

### 3.2 Using Peralta’s Scheme with Practice Ballots

The introduction of practice ballots requires reworking the privacy scheme. In Peralta’s scheme, the ballot is signed along with a random identifier in order to ensure uniqueness. However, the vote buying scheme requires that the government be able to distinguish between practice ballots and real ballots, by using the voter’s credentials. What is needed is a way to embed one bit of information into the random identifier, so that the government can determine whether the ballot is real (1) or practice (0). The mathematics of quadratic residues provides such a mechanism. When user identifiers are distributed, the identifiers for real ballots are quadratic residues, while those for practice ballots are quadratic non-residues. The protocol that follows allows the residuosity of the identifier to be preserved, while completely hiding the original identifier. It relies on the assumption that the voter has already logged in, and is communicating over a secure channel.

Conceptually, the protocol looks like this:

1. Voter chooses $k$ random blinding factors, and $k$ random powers. Voter computes voting tokens by raising her identifier to random odd powers. Voter then hashes and blinds these tokens and sends them to the Ballot Issuing Authority for signature.

2. Ballot Issuing Authority (BIA) chooses $k/2$ of the blinded tokens to challenge.

3. Voter responds to challenge by sending to government the blinding factors and powers that correspond to the challenged tokens. BIA verifies that the challenged tokens were properly constructed.
4. BIA signs remaining unchallenged tokens as well as their product and sends them back to voter.

5. Voter unblinds the signed, unchallenged tokens and the product.

The tokens and their product will serve as a set of anonymous credentials. They are sent in via an anonymous channel to complete the protocol:

1. Voter sends in ballot and voting tokens with their signatures, as well as the signature of their product to Vote Counting Authority (VCA). VCA verifies signatures, and verifies that the product has not been submitted before.

2. VCA computes the Legendre symbol modulo $p$ and $q$ for each token. The ballot is rejected if they are not all equal. If all are 1, the ballot is accepted as a valid real ballot and the votes are recorded as real. If all are $-1$, the ballot is accepted as a valid practice ballot and the votes are recorded as practice.

See Figure 1 for the precise mathematics of the first stage of the protocol. $u$ is the voter's identifier, $d$ is the private signing key, $e$ is the public verification key, $n = pq$ is their modulus, $S(x)$ is the RSA signing function using $d$, $H(x)$ is a one-way hash function and $k$ is the number of tokens in the original exchange, and is a constant of the protocol. At the completion of this protocol, the voter has $t_i$ which will serve as the voting tokens as well as their signatures $s_i$ and the signature of their product, $x$. Finally, $b$ is the voter’s ballot. They are then sent in via the anonymous protocol specified in Figure 2.

3.3 Why this Protocol Meets Its Goals

First of all, this protocol, assuming that the second portion is performed anonymously, ensures that the voter cannot be tied to the ballot; that is, the government cannot tell for whom a voter cast her votes. This property holds because the ballot is only submitted at the end, through the anonymous channel, along with voting tokens and a product. The government has signed these tokens in the earlier protocol without having seen them since they were blinded. The government cannot tie the tokens to the voter’s identifier because the identifier was raised to random powers, which the government also did not know.

Second, this protocol allows the government to distinguish real ballots from fake ballots. For real ballots, users receive identifiers which are quadratic residues, and for fake ballots, they receive identifiers which are quadratic non-residues. Raising these identifiers to odd powers to create the voting tokens does not change the
Voter

1. Choose random $p_i, r_i \in \mathbb{Z}_n^*$ for $1 \leq i \leq k$
   Compute $t_i = u^{p_i} \mod n$
   $z_i = r_i^e H(t_i) \mod n,$

Ballot Issuing Authority

2. Let $C = \{\text{size k/2 subset of } \{1, ..., k\}\}$
   and $\overline{C} = C \setminus \{1, ..., k\}$

3. Verify $z_i = r_i^e H(u^{p_i}) \mod n$ for $i \in C$

4. Compute $S(z_i)$ for $i \in \overline{C}$ and $y = S(\prod_{i \in \overline{C}} z_i) \mod n$

5. Compute $s_i = r_i^{-1} S(z_i) \mod n$
   and $x = (\prod_{i \in \overline{C}} r_i^{-1}) y \mod n.$
   Verify $s_i = S(t_i)$ and $x = S(\prod_{i \in \overline{C}} t_i)$

Figure 1: Mathematics of Protocol with Ballot Issuing Authority

Voter

1. Send in ballot, token pairs, signatures, and product.

Vote Counting Authority

$b, t_i, s_i, x$ for $i \in \overline{C}$

Verify $s_i = S(t_i), x = S(\prod_{i \in \overline{C}} t_i).$ Check that $x$ has not been used previously.

2. Check that $(\frac{t_i}{p}) = (\frac{t_i}{q}) = (\frac{t_j}{p}) = (\frac{t_j}{q})$
   for all $i, j.$ If all are 1, the ballot is accepted as a valid real ballot. If $-1,$
   the ballot is accepted as a valid practice ballot. If all are not equal, the ballot is rejected.

Figure 2: Mathematics of Protocol with Vote Counting Authority
residuosity; so if the identifier was a quadratic residue, all of the voting tokens would be quadratic residues as well, and if it was a quadratic non-residue, all of the voting tokens would be quadratic non-residues as well.

Third, voters and vote buyers cannot distinguish real ballots from practice ballots. Elements that are quadratic residues mod $n$, (which are therefore also quadratic residues mod $p$ and mod $q$) are indistinguishable from quadratic non-residues which are quadratic non-residues mod $p$ and $q$ unless one knows $p$ or $q$. Determining $p$ or $q$ requires knowledge of the factorization of $n$, which is exactly the secret maintained for RSA signing. Only the government, which knows the factorization of $n$, can determine whether the ballots are real or practice.

3.4 Resistance to Attacks by Dishonest Users

Breaking this protocol would involve substituting a practice ballot for a real ballot. In order to do this, a voter must submit $k/2$ signed voter tokens, all of which are quadratic residues mod $n$. I will show why all such attacks have an arbitrarily low probability of success.

One possible attack would be to send in quadratic residues as the voting tokens instead of generating the tokens by raising the identifier to odd powers. This attack would fail when the $k/2$ tokens are challenged, as he would not be able to produce proper odd powers and blinding factors to generate the random tokens he generated. An attacker could attempt to guess which tokens would be challenged, and generate those in the correct way (by raising the identifier to an odd power), and the others could be quadratic residues generated by the attacker at random. The attacker cannot generate more than $k/2$ tokens in the correct way, because then the Legendre symbols calculated in the second portion of the protocol will not all be equal (the correctly generated ones will be -1, and the attacker needs all to be 1). So, the attacker would have to guess exactly which set of $k/2$ the government will challenge. There are $\binom{k}{k/2}$ possible sets, so the probability of guessing the set that the government will challenge is $1/\binom{k}{k/2}$. At $k = 128$, this is approximately $1/2.4 \times 10^{37}$, or about a factor of 10 greater than the probability of guessing a 128-bit symmetric encryption key.

A second possible attack would be to choose special blinding factors so that the token would appear as if it was generated properly if challenged, but be a quadratic residue. In other words, the voter could have two blinding factors $r_1$ and $r_2$ such that $r_1^e H(u^{p_1}) = r_2^e H(u^{p_2})$ where $p_1$ is odd and $p_2$ is even. Solving for $r_2$, we get $r_2^e = r_1^e H(u^{p_1}) (H(u^{p_2}))^{-1}$. So the voter would need to solve the problem $r_1^e = k$, which amounts to determining the signature on $k$. Thus, completing this attack amounts to cracking an RSA signature, a problem which is believed to be hard.
4 Reference Implementation

“In theory, there is no difference between theory and practice. But, in practice, there is.” – Jan L. A. van de Snepscheut

In order to test the feasibility of this protocol, I created an implementation of the above theory. My two major goals in the creation of this implementation were security and simplicity. The reasons for security are obvious, and I think it is important that the interface of a voting system be simple and easy to use. An election system which prevents undercounting is worthless if voters cannot understand how to use it. As such, while security is the goal for the activities under the hood, simplicity is the focus for user interaction with the application. I tried to design an application with as few buttons and options as possible, and as little room for user error as possible.

4.1 Code

One of the decisions I made early on in the development of this application was to use Java as the programming language for development. Java has three main advantages. First, it is a cross-platform language. There are available distributions for computers running the Windows, Macintosh, Linux, and Solaris operating systems, and Java is the de facto cross-platform programming language. Although the majority of internet users browse the web using a Windows system, an internet voting system is most useful if it has as few requirements as possible. Using Java allows any user with a computer and internet connection to vote.

Second, Java is a memory-safe, type-safe language. As Professor Avi Rubin points out in his analysis of the Diebold code, “Furthermore, when building software that’s meant to be robust against attacks, one of the most painful lessons of the past decade of computer science has been the damage that can be caused when a program is vulnerable to memory-corruption and type-confusion attacks” [3]. As Rubin points out, type- and memory-safe languages, although by no means bug free, do “guarantee that a program’s operation is predictable, and many important classes of bugs, including buffer overflows, can be guaranteed to never occur” [3]. Java is a type-safe, memory-safe language, and thus is suitable for this kind of application in which reliability and robustness is crucial.

Finally, Java is the flagship language of the open-source movement. One of the biggest criticisms of the Diebold system and other electronic voting systems has been that the source code has been kept a proprietary secret. As has been demonstrated repeatedly throughout the history of cryptography, security based on the secrecy of an algorithm is easily broken. Allowing code to be open-source allows for review by
security and software experts across the internet, which in the end makes for a better product. Hopefully, the open-source product would allow any security flaws in the code to be discovered long before it is used in any election.

4.2 Logon and Authentication

The logon screen, in keeping with the simplicity concept, contains three text boxes for logon information and two buttons. The text boxes are for the entry of the user ID, password, and authenticator, which the user has received prior to voting. Each is a combination of capital letters and numbers. I decided to use only capital letters so that there would be no confusion over case sensitivity. Each field is divided into groups of 4 by dashes for easier entry. The user ID is 12 characters long, and the password and authenticator are each 8 characters long. Since each character can be either one of 26 letters, or one of 10 digits, there are 36 possible options for each of these logons per character, which comes out to be $4 \times 10^{19}$ different possibilities for user IDs, and $3 \times 10^{13}$ possibilities for passwords.

The logon protocol relies on each of the above fields (user ID, password, authenticator) being a shared secret, unique to each user. In an actual election, this could be achieved by mailing proper credentials to registered voters, or by distributing them at local town halls or schools. The protocol is fairly straightforward:

1. Voter sends the user ID to the server over an insecure channel.

2. Government generates 20-bit Message Authentication Code (MAC) for its RSA public encryption key using authenticator as MAC key. Government sends to voter its public key and MAC.

3. Voter computes MAC independently, and verifies that it matches the one sent by the government.

4. Voter encrypts random 128-bit symmetric AES session key with government’s public key and sends it to server. This key is used (with AES encryption) to encrypt the rest of the session.

5. Voter sends his password (encrypted) to the server.

6. Government checks password against its list. If it matches, the user is authenticated.

Clearly, the security of this process relies on the password and authentication key. The password assures the server that it is talking to the actual voter, and not...
just a voter who randomly guessed the correct user ID. The probability of guessing the correct password is extremely low, especially since the passwords are randomly generated, rendering dictionary attacks impossible.

The MAC assures the voter that he is talking to the official government server. The attacker’s goal would be to spoof the official government server by sending in his own public key, thereby obtaining the voter’s password. He could then use the credentials for himself to vote on the actual server. To prevent this, a MAC is computed for the government server’s public key. Anyone who knows the key can compute the same MAC and verify the authenticity of the message. Without knowing the key, however, the attacker cannot verify the message, and more importantly, cannot compute a MAC for a different message (in this case, a different public key).

In an ideal world, a MAC key should be at least 128-bits, however, using a 128-bit key would require excessive data entry on the part of the voter. Using a 40-bit key with a 128-bit MAC would subject the authentication to a simple brute force attack; an imposter could simply try all of the possible keys until he found the correct one, then use it to compute a MAC for his own public key. To prevent this attack, a 20-bit MAC is sent because this creates a code space which is half the size of the key space. Having a smaller code space than key space (assuming some nice properties of the MAC function) allows many possible keys to generate the same MAC for a given message. Therefore, an attacker can gather only some information on the key by seeing the MAC for the government’s public key. If he wants to compute the MAC for a different public key, he will not know which key to use. Setting the MAC length at 20 bits ensures that an attacker only gains 20 bits of information about the authentication key, leaving 20 bits worth of secret information. Since the MAC itself is only 20 bits long, the attacker has just as good a chance of guessing the MAC for his own public key as trying to compute it. At 20 bits, this chance is about 1 in a million.

I should also say a quick word about the generation of login credentials. The generation of practice and real logins is accomplished by the same means, but practice logins are generated once the election has begun, while real logins are generated before this date. In either case, the password and authentication key are generated randomly. Real logins are generated by squaring a random group element (creating a quadratic residue) and practice logins are randomly generated and then tested for the right quadratic residuosity.

4.3 Ballots

In order to prevent confusion whenever possible, the ballot screen displays one candidate per page. It is able to handle elections in which the voter may choose just one candidate, and those in which the voter may choose a larger number of candidates.
To prevent undercounting, the voter must choose a candidate or abstain before she moves on to the next candidate. When a voter has finished her selections, she reaches a confirmation screen which lists all the candidates she chose earlier. She can either go back and change her votes or click the vote button to submit her ballot. The ballot submission procedure follows the procedure described in Section 2.

## 4.4 Database

The database for the reference implementation is running Microsoft SQL Server, though this database was chosen out of convenience. The server can be reconfigured to use any database with JDBC drivers by changing only a few lines of code. Database contact is made only through stored procedures. This allows for maximum security, because arbitrary changes cannot be made and because it is easier to track changes. Also, in case changes have to be made in response to a change of database, they can be made in the database and not in the application server.

## 4.5 Cryptography

Most of the cryptography used in the reference implementation makes use of the Java Cryptography Extensions (JCE), a standard library included with Java v1.4. The algorithms used from JCE include AES for symmetric encryption, the HMAC-SHA1 keyed hashing algorithm for MAC generation, SHA-1 for hashing, and SHA-1 Pseudo-Random Number Generator for random number generation. RSA keypairs are also generated using the extensions; however, the RSA encryption and signing implementation was written for this application, as it is not included in the JCE [7].

## 5 A Comparison to SERVE

Although to date no local governments have instituted internet voting, the Department of Defense has worked on the creation of an internet voting system to enable American personnel abroad to vote when tangible absentee ballots are not a feasible option. It created SERVE, the Secure Electronic Registration and Voting Experiment. However, the system was ultimately not a success. In January 2004, four security experts who analyzed the system subsequently published a report detailing a number of security flaws. Soon after the publication of the report, the Department of Defense decided not to use the system for the 2004 general elections.[6]

The experts’ analysis of the SERVE system is interesting for two main reasons. First, SERVE was to be the first system in the United States used for large scale voting for general elections. Whereas earlier reports had concentrated on theoretical
systems, the January 2004 report analyzed an extant system that could be tested in a concrete way. Second, unlike reports on Diebold and other electronic voting systems, the analysis of SERVE specifically mentioned that proper procedures were followed to secure the system.

“We mean no criticism of the FVAP, or of Accenture, or any of its personnel or subcontractors. They have been completely aware all along of the security problems we describe here, and we have been impressed with the engineering sophistication and skill they have devoted to attempts to ameliorate or eliminate them. We do not believe that a differently constituted project could do any better job than the current team. The real barrier to success is not a lack of vision, skill, resources, or dedication; it is the fact that, given the current Internet and PC security technology, and the goal of a secure, all-electronic remote voting system, the FVAP has taken on an essentially impossible task. There really is no good way to build such a voting system without a radical change in overall architecture of the Internet and the PC, or some unforeseen security breakthrough” [1]

It is clear from this paragraph that the experts performing the analysis see the problem as with the medium, not the methodology.

Aside from vote buying and ballot privacy, the SERVE analysis mentioned a number of other vulnerabilities. I would like to discuss the feasibility of those attacks on the system as proposed and constructed earlier.

5.1 Man-in-the-Middle and Spoof Attacks

The SERVE security analysis devotes an entire section to the problem of spoofing and man-in-the-middle attacks. The victim of such an attack can be disenfranchised, or even have her vote changed. SERVE uses SSL to authenticate the government’s servers. Unfortunately, any dishonest certificate authority can give a certificate to an attacker. Internet Explorer comes pre-installed with a number of root-level certific-ates, some of which are from companies not based in the United States. As a result, the use of SSL opens up the possibility of a foreign company rigging elections in the United States. On the other hand, the proposed system uses a different authentication protocol based on shared authentication keys. This system avoids reliance on outside certificate authorities, relying instead on a shared secret between the government and the voter. It is also stronger than a system in which the government’s public key is hard-coded in the application, because it allows for a new key pair to be used if an old key pair is compromised, or for different government servers to use different public keys.
5.2 Control of the User Environment

The SERVE report characterizes a number of different security problems as resulting from lack of control of the voting environment. In order to add some more control to the environment, I suggest that the voting application be sent to voters on a CD. First, this prevents a virus from modifying a downloaded application, as it would be in a read-only location. Certainly, it would still be possible for a virus to affect the application, but at least this would make it more difficult. As a later, more drastic measure, I suggest that the CD should be bootable. Perhaps it could be loaded with a simplified version of an OS like Knoppix, which can operate fully while running only from a CD. The CD would only need to contain enough of an OS to run the application, and connect to the internet.

At this time, it is impossible to use such a system for every voter. First, Knoppix is not simple enough to be used by the average voter. Furthermore, as most voters connect to the internet via dial-up connections, they often use specialized connection applications. However, the trend is moving toward broadband connections, so perhaps a simplified, easy-to-use version of the Linux operating system with support for just Ethernet and the voting client can be developed. Furthermore, setting up clean workstation kiosks is certainly a possibility. Having special voting kiosks at standard voting sites would still allow some of the benefits of internet voting, such as accessibility for the disabled, and prevention of undercounting, while at the same time providing an inexpensive voting solution to local precincts, as it only requires standard off-the-shelf computers.

5.3 Server Attacks

The SERVE report also mentions the possibility of an attack on the central election servers. Clearly, this would be a serious concern for any voting system. In order to help minimize the risk of such an attack, all of the voting hardware must be kept behind a firewall. Furthermore, the server should be written in a type-safe, memory-safe language to prevent buffer overrun vulnerabilities. But in general, the requirements of the voting server are not terribly different from those of a high-security e-commerce server, such as a bank server. In some sense, it would be a similar blow to the United States if a major financial institution’s server was compromised and money was redistributed to different accounts as it would be if the voting server was compromised and votes were redistributed. Clearly, crucial systems must be protected with strong security measures, and be tested with malicious, improper inputs (they must fail gracefully, and not allow a hacker to execute malicious code). But this is not beyond the realm of current security practices; in this area, the voting server does not seem so different from the e-commerce servers so prevalent throughout the
5.4 Denial of Service

Perhaps the most serious vulnerability is not really a security vulnerability at all. A denial of service attack against the voting servers could disenfranchise large numbers of voters. As pointed out in the SERVE analysis, extending the voting window does not solve the problem, as an attack close to the deadline would still be crippling. The possibility of a denial of service attack is not purely theoretical either. In recent years there have been attacks on a number of major websites such as CNN, Yahoo, and eBay as well as the White House. Finally, and perhaps most telling, an internet election in Canada was disrupted by such an attack [1]. Large scale use of internet voting for national elections would create the best target yet for a denial of service attack. The problem can be mitigated by having an excessive amount of computing resources. By having extremely large bandwidth capabilities and an enormous amount of server power, mounting an effective denial of service attack can be made more difficult, but not impossible.

5.5 Insider Attack

Any kind of internet voting system would also be vulnerable to an insider attack. A database or system administrator would have the power to affect the election in any number of ways. He could shut down an election, alter votes in the database, or add or remove logins. However, such an insider attack is not really a problem unique to internet voting. In a traditional system, an insider could rig mechanical voting machines or optical readers to incorrectly count votes, or add or remove ballots at the final counting location. In the traditional system, there are procedures, however, to try to avoid such problems. The administration of an election is done with multiple observers present at all stages, so that no individual can single-handedly affect the election. The same should be done for any sort of internet election. Procedures should be in place such that the installation of all software on the server is done openly, with monitors, with no one person having access to the database or to the server. In truth, any centrally administered voting system is vulnerable to an insider attack, but by setting up proper administrative procedures, we can minimize the risk.

5.6 Lack of Audit Trail

Interestingly, much of the computer science community has rallied around the cry for paper audit trails for electronic voting. They see a major flaw of the current DRE (direct recording electronic) machines to be the lack of such a trail, and the
SERVE analysis makes a similar criticism of internet voting. However, I think that the strength of this movement towards audit trails comes largely from the current state of affairs of electronic voting manufacturers. Systems from Diebold and others have been demonstrated to be highly insecure; accordingly, proponents of the audit trail see it as a way to prevent flaws from affecting the final tally. However, an audit trail is not the universal elixir; it is not even necessarily helpful. In an open-source voting system, voters can be assured that the system is acting properly at both the client and server end, and encryption mechanisms can assure that this connection is proper.

Furthermore, the introduction of an audit trail creates new problems. In a hypothetical close election, the count from the electronic or internet machines could differ from the count from the paper audit trail by enough that each indicates a different winner. Now, we would be faced with the problem of determining which tally is correct. While it is possible that the electronic machines were tampered with, it is also possible that the slips of paper were tampered with or just simply lost. A joint MIT-Caltech paper on voting cites the case of a poll worker who accidentally brought home a bag of ballots, confusing it with his laundry [4]. There are also a number of ways in which an insider could modify the audit trail just as easily if not more easily than he could modify the electronic count. For these reasons, I don’t believe that the lack of an audit trail should cause us to completely give up on the idea of internet voting altogether. It does, however, mean that we need to be even more careful about security and tamper prevention.

5.7 Mail Interception

In the system I propose here, there is also a new vulnerability introduced, one that was not even present in the SERVE system which uses online registration. An attacker could simply intercept the mailings of credentials in order to disenfranchise voters. He could vote multiple times by using stolen credentials, and even give voters practice logins, so that they would not even know that they were being disenfranchised. While this is certainly a problem, I do not believe it warrants eliminating the entire system. Voters in Oregon do all of their voting by mail so they are already subject to such an attack, and all other states have some kind of absentee balloting system by mail. Mail interception, while serious, is not really a disadvantage of internet voting, but of absentee voting altogether. As I said from the beginning, it is not fair to evaluate internet voting security against perfection; it should strive to be at least as good as what we have now.

Furthermore, there are defenses against this type of an attack. Practice logins could be disabled until the start of the election, at which point a voter would know if she had not yet received credentials. Voter IDs could be recorded by name and
address; if a voter has not yet received credentials, the government could cancel her existing credentials in the same way that a lost credit card is cancelled and issue her new ones. Obviously, each of these defenses introduces new vulnerabilities (privacy in the first case, insider attack in the second), but with proper procedure, I think these vulnerabilities can be reduced.

5.8 Digital Divide

An interesting problem with internet voting is the problem of selective disenfranchise-ment created by the digital divide. Home computers with internet connections are still more prevalent in younger households than in older households, wealthier households than in poorer households, and white households than in households of color. While clearly there should be public computers available for voting in traditional polling spots, there would almost certainly be higher turnout among those who have home internet access at their finger tips. Some critics of internet voting, including the Reverend Al Sharpton have even said that internet voting is unconstitutional because it amounts to a poll tax [5]. While this is more of a policy issue than a security issue, it is certainly worth discussing, and not overlooking the socio-economic implications of allowing home internet voting.

6 Conclusions

Since the 2000 election fiasco in Florida, there has been a push to update the nation’s voting technologies. We became aware of the grave problems with our nation’s voting systems, and were appalled by the number of votes lost or not counted. In a country where every vote is supposed to count, 4 to 6 million votes were lost in an election decided by a few thousand [4].

To the general public, the obvious solution to this voting crisis seems to be the use of computers. Numerous counties nationwide have purchased new electronic voting systems, many of which have been widely criticized by computer scientists and security experts for their insecurity and poor development practice. Of particular note is Diebold, whose proprietary, secret code was found unprotected on the internet. Researchers from Johns Hopkins performed a detailed analysis of the system, and noted numerous security vulnerabilities. However, the push for electronic voting continued.

With the prevalence of the internet in today’s society, the use of the internet seems to be the next logical step. People are used to conducting bank transactions and purchasing goods online, and do not see the crucial differences between e-commerce and voting. Despite the ambivalent attitudes of programmers, social activists, and
the everyman, the march toward internet voting is likely to continue. While use of
the SERVE system was canceled for the 2004 elections, it is not likely to be the
last attempt at an internet voting system by the Department of Defense or by other
government entities. We must do more than simply criticize systems for their vulner-
abilities. We must work to develop solutions. Hopefully, the system proposed in this
paper helps address some of these issues.

References

[1] Jefferson, Dr. David, Dr. Aviel D. Rubin, Dr. Barbara Simons, and Dr. David
Wagner. A Security Analysis of the Secure Electronic Registration and Voting


http://avirubin.com/vote/response.html


Times 6 Feb. 2004

http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html

[8] Peralta, Rene. Issues, non-issues, and cryptographic tools for internet-based vot-
ing, Chapter 11 in Gritzalis D. (Ed.), Secure Electronic Voting, Kluwer Academic