Internet Voting Protocol Based on Improved Implicit Security

Abhishek Parakh and Subhash Kak

Computer Science Department, Oklahoma State University
Stillwater, OK 74078

September 11, 2009

Abstract

This paper presents an improved protocol for Internet voting that allows recasting of ballots to eliminate voting errors, encourages early voting and provides an opportunity for changing votes as the election campaign progresses. The protocol is able to meet the competing requirements of verifiability and privacy by using a distributed security architecture, involving multiple servers, and multiple audit channels. It also detects servers that might be cheating.

Keywords: Internet voting protocol; distributed data security; ballot recasting.

1 Introduction

The contrasting requirements of confidentiality and verifiability are fundamental to many problems of theoretical computer science. The requirements are such that they preclude a general solution but satisfactory solutions are possible for specific applications. Internet voting represents an application area where these requirements may be met using tools of cryptography and a distributed security architecture. In spite of certain limitations of current methods [17], Internet voting is expected to be used widely in the next few years [8] since it offers ease of access to senior citizens, disabled people, people traveling on election-day, citizens living abroad who are eligible to vote, and eliminates the hassle of obtaining an absentee ballot in advance. It also encourages larger participation on the part of the younger generation that has become accustomed to online banking, online shopping, secure email transactions and secure online storage. The acceptance of Internet voting will become wider if limitations of the current methods are lessened.

Several voting schemes are to be found in the literature [1, 3, 10–14, 17]. Conventionally, blind signatures [7] are popular for achieving ballot secrecy,

while anonymization is performed using Mix-Nets that use multiple encryptions, decryptions and random permutations [10–12]. Other schemes treat votes as e-cash [1–3], exposing the voter and his ballot if a recast is attempted. Some schemes propose to protect intermediate election results [14] by asking the voter to encrypt the vote. However, it is not clear how the decryption key is to be provided at the time of vote counting. Often postal services are used as a part of the scheme to provide untappable channels [13], but their use would clearly defeat the purpose of Internet voting that aims at eliminating postal delays and paperwork. Further, such schemes are not suitable for large scale elections due to the overhead.

In order to minimize the effects of viruses, Trojan horses and denial of service attacks [9], Internet elections are held over a period of several days, as in the case of Switzerland for two weeks [5], and in Estonia [6] for a few days. Despite the long voting periods, undecided voters and conservative people tend to wait until the last day to cast their ballots which causes the network attacks to be aimed at this last minute voting period. Further, proper design of ballots is a critical element in their effectiveness. Poorly designed interfaces and ballots can lead to voting errors (similar to errors in choosing wrong options in online bookings and purchases), with the unacceptable effect of a wrong candidate receiving the vote. Instances of such mistakes were seen in the California Recall elections [24]. Most existing protocols do not make any provisions to correct such mistakes.

One could propose that voting be allowed only after the campaigning for the elections is over. But to ensure that campaigning cease on a certain date might be impossible to enforce, and the public information available on the media can lead people to change their mind. Further, such an idea would not eliminate mistakes in voting.

Therefore, we believe that election systems should be designed to allow ballot recasts, i.e. a voter should be able to change his vote either during the period that the voting is open or during a specified time frame when ballot recasts are allowed. This will encourage a larger participation by voters in the early voting process because they will now have an option of changing their votes if they wished to do so. We further believe that ballot recasts will act to deter vote selling, because the vote buyer cannot be sure if the seller has sold his vote to multiple parties.

The scheme described in this paper allows voters to recast ballots, while protecting the intermediate election results using an implicit data security architecture. Rabin [16] spoke of distributed storage of data over different servers in order to protect the data. He, however, discounted the use of secret sharing schemes for data distribution because of their space inefficiency and resorted to an insecure method of distribution. However, since then several space efficient secret sharing schemes [18–21] have been developed based on the idea of recursion. In recursive secret sharing, the original secret is broken into several pieces and the pieces are divided into shares one by one, by repeatedly using already generated shares. Since we only have a small secret, namely the cast ballot, to share between different servers, in this paper we use recursion to hide additional

information within the shares to provide a mechanism for validation of shares. We call this model of security as implicit data security, where the word implicit signifies that no explicit encryption keys have been used, but the data is stored in a manner that every piece is implicitly secure.

The problem of recast of ballots was considered in an earlier scheme by the authors [22], in which the intermediate election results are protected using secret sharing. However, the scheme has three shortcomings: the lack of redundancy in the system, where the loss of a single piece of the ballot results in a loss of that vote; lack of verifiability for the pieces brought together during the tallying process; and there is no mechanism for cheater identification.

2 Our Contribution

The article aims at presenting a scheme that overcomes the limitation of our earlier scheme [22]. In the previous scheme we use the technique of generating anonymous IDs for ballot secrecy that eliminates the need for MixNets. These anonymous IDs then become the anchor point for the votes and are recorded along with the cast ballot. Consequently, to change a vote, the voter only needs to send the anonymous ID and the new ballot, which replaces the old one in the system. Our contribution here is threefold:

First, we propose a new recursive method for dividing the votes into n pieces such that loss of some of the pieces (less than a threshold k) does not result in the loss of the vote.

Second, we present a method for validation/verification of the ballot pieces that are brought together at the time of tally.

Third, a variant of the protocol is presented that can detect invalid ballot pieces as well as expose the cheating server.

The improvements are implemented using recursion such that it does not lead to an increase in share size and stores only a small constant amount $O(1)$ of information for verification.

3 The Proposed Protocol

The protocol proceeds as follows: The voter contacts the registration authority (RA) using his real identity. He then randomly generates a number, blinds it and gets it signed by the RA. Upon receiving this signed number, he unblinds it and uses it as his anonymous ID. Consequently, to cast his ballot, the voter does not blind the ballot but simply uses the anonymous ID to vote. However, in order to protect the intermediate election results, the voter divides his ballot into several pieces and sends them to distinct servers. This provides a distributed security architecture. Before dividing the ballot, the voter encodes certain secret information within the pieces of the ballot and distributes a one-way hash of the secret information among the servers with the ballot pieces.

We present a new procedure for dividing the votes into pieces such that additional information may be hidden within the pieces. This hidden information can be used to validate the pieces at the time of ballot counting. Further, the additional information does not increase the share sizes and does not increase algorithmic complexity of the scheme being used.

The procedure for registration and creating anonymous ID is similar to that described in [22]. Here, g^x is the public key of the registration authority (RA) and x is its corresponding private key. All computations are performed modulo a prime p where g is a primitive root that is taken to be public knowledge. Further, m^x denotes the signature on a message m, where $1 < m < (p-1)$. A one-way hashing function $h(\cdot)$, used for verification, satisfies following properties [23].

1. $h(\cdot)$ when applied to an argument produces a fixed-size output.
2. Given x, it is easy to compute $h(x)$; but given $h(x)$ it is computationally infeasible to determine x.
3. $h(\cdot)$ is collision free, i.e. it is computationally infeasible to find distinct x and y with $h(x) = h(y)$.

At the time of registration the voters send a one-way hash of the anonymous ID that they have generated. The registration authority maintains a list L of all the one-ways hashes that have been used in order to avoid collisions between the anonymous IDs.

It is assumed that the voter has n servers at disposal to cast his ballot; and any k of them are required to reconstruct the ballot, $k \leq n$. Every vote V is assumed to be a number in $\mathbb{Z}_p$ that is assigned to a candidate and is public knowledge.

3.1 Registration Phase

1. The voter randomly and uniformly chooses a number $r_{id} \in \mathbb{Z}_p$.
2. The voter randomly and uniformly picks a blinding factor b and computes $u = r_{id} \cdot g^b \bmod p$. He sends a 3-tuple to the registration authority (RA): u, V_{id}, $h(r_{id})$; where V_{id} is his true identity in clear-text.
3. If the registration authority finds the voter eligible to vote (by checking his V_{id}, it checks if $h(r_{id})$ is already present in L. If the hash is found, then the RA requests the voter to repeat steps 1 & 2. If the hash is not found, the RA adds the new $h(r_{id})$ to L and sends to the voter u^x.
4. The voter retrieves the signed r_{id} as follows:
   (a) Using the RA's public key compute $v = (g^x)^b \bmod p$ and v^{-1}.
   (b) Compute $r = u \cdot v^{-1} = (r_{id})^x \bmod p$.
5. The voter randomly and uniformly chooses an exponent c and generates: $d = ((r_{id})^x g^x)^c$ and sends it to the RA.

6. The RA sends back: $e = d^{\frac{1}{x}}$.

7. The voter compares if $(r_{id} \cdot g)^c$ is equal to e, he accepts $(r_{id})^x$ as valid signature.

The voter at the end of registration phase has a valid signed r_{id} that may be used to vote and proceeds to cast his ballot as follows.

3.2 Voting Phase

1. The voter contacts the online polling station using a secure shell (SSL) connection over the Internet and sends: $(r_{id})^x \bmod p$, r_{id}.

2. The polling station verifies the validity of r_{id} by conducting a zero-knowledge challenge/response protocol with the RA to validate the signature.

3. If the signature is found valid the online polling station stores the voter's r_{id} and provides the voter with a secure session key that he can use to cast his ballot by contacting the n voting servers.

4. The voter generates a random secret message M.

5. The voter chooses the ballot V corresponding to the candidate that he wants to vote for and divides the ballot into n pieces using `Algorithm A(V,M,k,n)` (detailed below) which returns a set of n points, $(j + k - 1, D_j)$, $\forall\ j$ from 1 to n.

6. The voter sends to every server S_j: $((j + k - 1, D_j), h(M))$, for all $j = 1$ to n.

`Algorithm A(V,M,k,n)` takes as input: ballot `V`; message `M`; and integers `k` and `n`.

`Algorithm A(V,M,k,n) - Dealing Phase`

1. Divide M into $k - 2$ pieces: m_1, m_2, ..., $m_{k-2} \in \mathbb{Z}_p$, such that concatenations of m_is, taken in order, is equal to M.

2. Choose randomly and uniformly a number r_1 and compute $r_2 = m_1 \cdot (r_1)^{-1} \bmod p$.

3. For $i = 2$ to $k - 2$.

   Compute $r_{i+1} = m_i \cdot (\Pi_{j=1}^{i} r_j)^{-1} \bmod p$.

4. Map r_i as points (i, r_i), $1 \leq i \leq k - 1$.

5. Map the ballot V as point $(0, V)$.

6. Interpolate points $(0, V)$ and (i, r_i), $1 \leq i \leq k - 1$ as the polynomial $f(x)$.

7. Evaluate n samples: $D_i = f(x)$, where $x = k, k+1, k+2, \ldots, (k+n-1)$.
8. Output $(i+k-1, D_i)$, $1 \leq i \leq n$.

To understand the working of Algorithm A, consider the following example.

Example 1. Let the voter have $n = 8$ servers at his disposal to cast his ballot. Each of these servers may be at separate locations and controlled by an independent authority. Let $k = 5$ be the threshold number of servers that must remain honest and come together to reconstruct a vote correctly. Further, let the prime field chosen to work in be $p = 257$. Each candidate on the ballot is assigned a candidate ID which may be a random number from the field $\mathbb{Z}_p$. Consequently, to cast a ballot the voter must send this number (the candidate ID) to the servers. Assume, in order to implement authentication the voter chooses to hide a secret message $M =$"USA", in his ballot. The message to be hidden is entirely a choice of the voter and is it may as well be "RUSSIA".

Let us assume that the voter wishes to cast a ballot to a candidate with candidate ID 157. Therefore, the cast ballot is $V = 157$. If we consider the voter's secret message $M =$"USA" to be formed of ASCII characters, then M can be divided into three pieces such that $m_1 = 85$, $m_2 = 83$ and $m_3 = 65$, such that each belongs to $\mathbb{Z}_p$. Under such conditions, the algorithm proceeds as follows,

1. Divide M into 3 pieces: $m_1 = 85$, $m_2 = 83$ and $m_3 = 65$.
2. Choose randomly and uniformly a number $r_1 = 101 \in \mathbb{Z}_p$ and compute $r_2 \equiv m_1 \cdot r_1^{-1} \equiv 85 \cdot (101)^{-1} \equiv 85 \cdot 28 \equiv 67 \bmod 257$.
3. Compute $r_3 \equiv m_2 \cdot (r_1 \cdot r_2)^{-1} \equiv 83 \cdot (101 \cdot 67)^{-1} \equiv 83 \cdot 127 \equiv 4 \bmod 257$.
4. Compute $r_4 \equiv m_3 \cdot (r_1 \cdot r_2 \cdot r_3)^{-1} \equiv 65 \cdot (101 \cdot 67 \cdot 4)^{-1} \equiv 65 \cdot 96 \equiv 72 \bmod 257$.
5. Map the r_is as points $(1, r_1) = (1, 101)$, $(2, r_2) = (2, 67)$, $(3, r_3) = (3, 4)$, and $(4, r_4) = (4, 72)$.
6. Map the ballot V as point $(0, V) = (0, 157)$.
7. Interpolate the five points, $(0, V)$ and (i, r_i), to generate a 4^{th} degree polynomial, $f(x) = 148x^4 + 3x^3 + 251x^2 + 56x + 157 \bmod 257$.
8. Evaluate $f(x)$ at eight points (starting from $x = k$): $D_1 = f(5) = 128$, $D_2 = f(6) = 240$, $D_3 = f(7) = 173$, $D_4 = f(8) = 160$, $D_5 = f(9) = 131$, $D_6 = f(10) = 227$, $D_7 = f(11) = 29$, and $D_8 = f(12) = 100$.
9. Return $(j+k-1, D_j)$s; where $k = 5$ and $j = 1$ to n.

Note that in the above example, the voter has recursively encoded the pieces of his secret message into the shares of the ballot. Steps 2-4 generate 4 points that are required to simulate the 4 randomly chosen points in the Shamir's secret sharing scheme. Step 7 executes Shamir's scheme assuming r_is are randomly chosen points and the ballot as the y-coordinate at $x = 0$. Finally, the algorithm returns 8 shares of the cast ballot, these shares have the secret message hidden within them. Each of these eight pieces is sent to a different voting server, along with the hash of the secret message.

3.3 Counting Phase

Assume a threshold k, number of servers pool their pieces, $(j + k - 1, D_j)$s, together to reconstruct the votes. (If more than the threshold number of servers are available, then it may provide a mechanism for cross-checking and verification.)

1. Use `Algorithm A - Reconstruction Phase` (detailed below) to reconstruct the hidden message and the polynomial $f'(x)$.
2. Concatenate m'_js in order to reconstruct M'.
3. Compute $h(M')$.
4. If $h(M')$ is equal to $h(M)$, then the shares are valid and $f(x) = f'(x)$.
5. Retrieve, the cast ballot as $V = f'(0)$.

`Algorithm A - Reconstruction Phase` takes as input a set of points (shares) and returns the pieces m'_j of the hidden message and a polynomial $f'(x)$ constructed by interpolation of the shares.

`Algorithm A - Reconstruction Phase`

1. Interpolate the shares (points) to reconstruct $f'(x)$.
2. Sample $f'(x)$ at points $x = 1, 2, \ldots, k - 1$ to retrieve r'_i, $1 \leq i \leq k - 1$.
3. Reconstruct the hidden message as follows:
   (a) Do for $j = 1$ to $k - 2$: $m'_j = \Pi_{i=1}^{i=j+1} r'_i \bmod p$.
4. Return m'_j, $1 \leq j \leq (k - 2)$, and $f'(x)$.

Example 2. Recall from example 1, we had created eight shares (or pieces) of the cast ballot and each of these pieces were sent to a different voting server. After the polling is closed, the servers combine their pieces to recreate the cast ballot. Since only $k = 5$ pieces are required for recreation, assume that the 5 pieces that are brought together for reconstruction of the vote are $(6, D_2) = (6, 240)$, $(7, D_3) = (7, 173)$, $(9, D_5) = (9, 131)$, $(11, D_7) = (11, 29)$, and $(12, D_8) = (12, 100)$. The reconstruction proceeds as follows,

1. Interpolate the shares to reconstruct the 4^{th} degree polynomial $f'(x) = 148x^4 + 3x^3 + 251x^2 + 56x + 157$.
2. Sample $f'(x)$ at $x = 0$ to retrieve the ballot $V' = f'(0)$. (We denote the retrieved ballot as V' because its validity is yet to checked, which is done in the following steps.)
3. Sample $f'(x)$ at 4 points $x = 1, 2, 3, 4$: $r'_1 = f'(1) = 101$, $r'_2 = f'(2) = 67$, $r'_3 = f'(3) = 4$, and $r'_4 = f'(4) = 72$.
4. Reconstruct the pieces of the hidden message,
   (a) $m'_1 = r'_1 \cdot r'_2 = 85$.
   (b) $m'_2 = r'_1 \cdot r'_2 \cdot r'_3 = 83$.
   (c) $m'_3 = r'_1 \cdot r'_2 \cdot r'_3 \cdot r'_4 = 65$.
5. Concatenate m'_1, m'_2, and m'_3 to reconstruct M'.
6. Evaluate the hash: $h(M')$.
7. If $h(M')$=$h(M)$, then the shares are valid and the vote, $V' = V$, is counted in the tally.

If more than 5 servers come together to recreate the ballot, then any 5 shares can be chosen at random and their validity checked. If the shares turn out to be invalid, then another set of 5 shares can be checked or the extra share can be used to cross check the reconstructed ballot. However, for detection of invalid pieces, only 5 shares are required.

3.4 Recasting of ballot

If a voter wishes to change his vote, he may do so by contacting the online polling booth using his r_{id} to authenticate himself and follow the procedure described above. The servers will overwrite his previously cast ballot partitions if any.

The protocol has been able to detect attempts of cheating by validating the pieces of the cast ballot.

4 Security of the proposed protocol

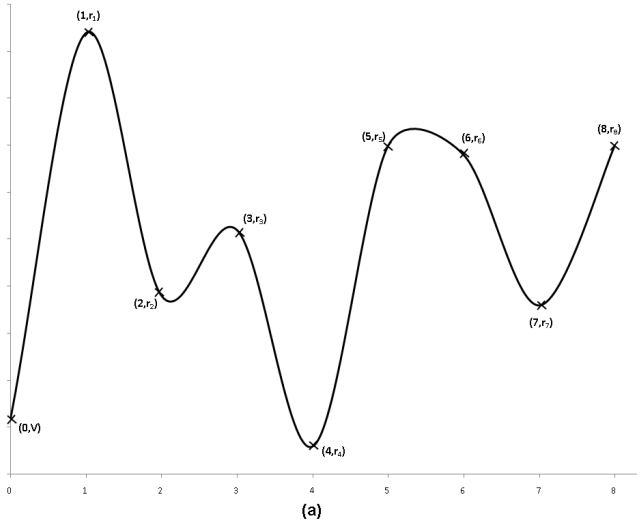
The security of the proposed protocol depends on the security of the pieces created for the votes. First, in the dealing process we recursively encode the secret message M. We also provide each server with $h(M)$. Hashing functions are known to be computationally secure and do not provide any information about the hidden message M, in practice. Since the message M is a randomly chosen secret message and r_1 was randomly and uniformly chosen from the

field, we may consider (i, r_i)s as random points. We use these random points to interpolate a polynomial with the vote as the coefficient free term (as a point mapped at $x = 0$). This polynomial is then sampled at n new points. If $k - 1$ servers collude to cheat, they would have no knowledge of the k^{th} point and by the properties of interpolation, all the polynomials will remain equally probable and therefore, all the ballots will remain equally probable, i.e. if there are m valid ballots, then each of them have a probability of $\frac{1}{m}$.

Now assume that an extremely powerful server is able to break the hashing function determining m_is. The next task, in order to determine the vote, is to determine r_js. But it turns out that knowledge of M does yield all the values of r_js as shown below: m_1 is chosen to be a product of two numbers r_1 and r_2, here r_1 is randomly and uniformly chosen from the field and therefore given any number y from the field $Pr(y = r_1) = Pr(y = r_2) = \frac{1}{p}$. Hence, the pieces of m_1 are secure. However, r_i, $3 \leq i \leq k - 1$ may be determined, using the m_is.

Further, the voting protocol uses r_is as points at $x = 1$ to $k - 1$ along with the ballot V at $x = 0$, i.e. V is the coefficient free term in the polynomial. It turns out that since we cannot determine r_1 and r_2 with a probability greater than a random guess, all polynomials remain equally probable, even if M is known. The last step of division of ballots into pieces, although not the same, is close to Shamir's implementation of secret sharing [15].

Figure 1. shows two graphs. Graph (a) illustrates the process of polynomial interpolation using $(0, V)$ and (i, r_i), $i = 1$ to 8. Graph (b) depicts the fact that if M is known (by breaking of the hashing function), then r_1 and r_2 remain undetermined and do not reveal any information about the ballot because all the polynomials remain equally probable; a few polynomials are shown for illustration.



The protocol further satisfies the various requirements of electronic voting:

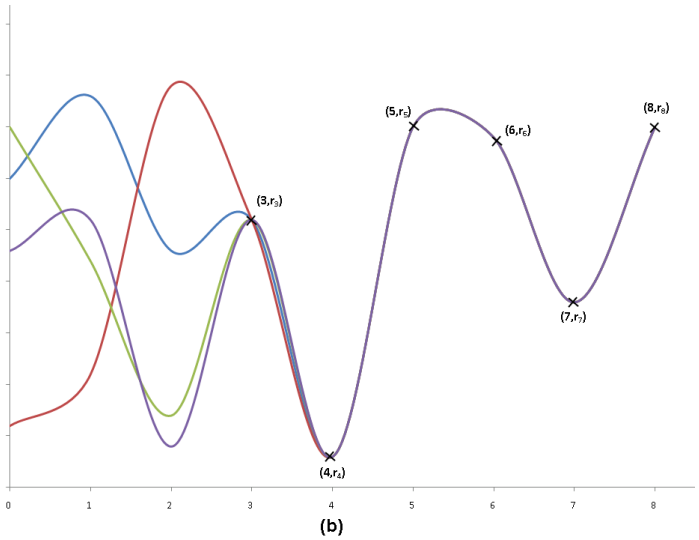


Figure 1: (a) Polynomial interpolation using vote $(0, V)$ and (i, r_i), $i = 1$ to 8. (b) Lack of knowledge of $(1, r_1)$ and $(2, r_2)$ leaves the polynomial undetermined; hence the vote remains secure.

1. Receipt freeness: The proposed protocol is receipt free.

2. Distributed security: It is assumed that at least one of the servers will be honest (an assumption similar to that of MixNet schemes) providing assurance to the voter that his vote will be counted as cast and any discrepancy will be detected with a very high probability.

3. Fairness: The ballot cast is divided into partitions during the voting phase and unless the partitions are known, all the ballots remain equally likely.

4. Un-forge-ability: An ineligible voter cannot cast a vote because the vote consists of two parts: the signed ballot and the certified r_{id}. The ineligible voter needs to generate a random ID and forge the signature of the registration authority, i.e. solve for x, given g, p and g^x mod p. This is equivalent to solving the discrete log problem, which is computationally infeasible. Leakage of information about the signature exponent is avoided by using a zero-knowledge challenge and response protocol for signature verification.

5. Un-reusability: Since every eligible voter is issued only one r_{id}, it cannot be used to cast multiple votes because if that is done the servers will simply overwrite the previous cast ballot for that r_{id}.

5 A Variation for Identification of Cheating Server/s

To identify the cheating server/s, we make use of the following properties of arithmetic coding (see [23]),

1. Let $T = \Sigma_{i=1}^{n} a_i p^{i-1}$, where $0 \leq a_i < p$, then
$$\lfloor \frac{T}{p^{j-1}} \rfloor \text{mod p} \equiv a_j$$
2. Let $T = \Sigma_{i=1}^{n} a_i p^{2(i-1)} + \Sigma_{i=1}^{n-1} c p^{2i-1}$, where $-p < a_i < p$ and $1 \leq c < p$, then
$$\lfloor \frac{T}{p^{2(j-1)}} \rfloor \text{mod p} \equiv a_j$$

Following minimal changes are required in the proposed protocol to implement cheater identification.

5.1 Voting Phase

We apply the voting phase as described before until step 5. Step 6 is replaced by the following 2 steps,

1. The voter computes
$T = \Sigma_{i=1}^{n} h(D_i) p^{2(i-1)} + \Sigma_{i=1}^{n-1} c p^{2i-1}$, where $c \in \mathbb{Z}_p$ is a random constant.
2. The voter sends to every server S_j: $((j+k-1, D_j),\ h(M),\ T)$, for all $j = 1$ to n.

Example 3. As an example to illustrate the operations described above, consider that $k = n = 3$ is the number of shares the voter has created for his ballot, given by $(1, D_1) = (1, 4)$, $(2, D_2) = (1, 2)$ and $(3, D_3) = (1, 5)$. Let $p = 11$ and let the voter choose a random number $c = 7$ from field $\mathbb{Z}_p$. Then before sending the shares of his ballot to the server the voter needs to compute $T = \Sigma_{i=1}^{n} h(D_i) p^{2(i-1)} + \Sigma_{i=1}^{n-1} c p^{2i-1}$.

For simplicity, consider a dummy hashing function with mapping $h(x) = x + 1$. Therefore, $h(D_1) = 4 + 1 = 5$, $h(D_2) = 2 + 1 = 3$ and $h(D_3) = 5 + 1 = 6$. Consequently, $T = 5 \cdot 11^{2 \cdot 0} + 3 \cdot 11^{2 \cdot 1} + 6 \cdot 11^{2 \cdot 2} + 7 \cdot (11^{2 \cdot 1 - 1} + 11^{2 \cdot 2 - 1}) = 97608$.

The voter now deposits this value of T on every server along with his ballot.

5.2 Cheating Server Identification

To identify a cheating server during reconstruction phase, following steps are performed,

1. Every server S_j presents their pieces $(j + k - 1, D'_j)$s, to evaluate $T' = \Sigma_{S_j} h(D'_j) p^{2(j-1)}$.
2. For each S_j, compute $c = \lfloor \frac{T - T'}{p^{2(j-1)}} \rfloor$ mod p.

3. Each server makes the above calculations and based on a majority of results, if $c = 0$, then S_j is honest, else S_j is cheating and its share is invalid.

We see that $c = 0$ if $\lfloor \frac{T}{p^{2(j-1)}} - \frac{T'}{p^{2(j-1)}} \rfloor = h(D_j) - h(D'_j) = 0 \bmod p$, which happens only when $D_j{=}D'_j$, i.e. the pieces presented by the server are valid pieces. An analysis of the identification protocol can be found in [23], where the solution has been applied to secret sharing.

Example 4. Continuing with example 3; assume that 3 servers have come together to reconstruct the ballot. Also assume that the reconstructed vote turns out to be invalid after checking the hash of the retrieved hidden message against the hash deposited by the voter (see example 2), then in order to determine which server is cheating, the servers perform the following.

Each server presents its share. Let us assume that sever 2 has cheated and provided an incorrect share such that $D'_2 = 3$. Servers 1 and 3 have provided correct shares $D'_1 = 4$ and $D'_3 = 5$. The servers then compute, $T' = \Sigma_{S_j} h(D'_j) p^{2(j-1)} = 5 \cdot 11^{2\cdot 0} + 4 \cdot 11^{2\cdot 1} + 6 \cdot 11^{2\cdot 2} = 88335$.

Now for each server S_j, compute $c = \lfloor \frac{T-T'}{p^{2(j-1)}} \rfloor$. Therefore, for server 1, $j = 1$, and $c = \lfloor \frac{97608-88335}{p^{2(1-1)}} \rfloor = 9273 \equiv 0 \bmod 11$. For server 2, $j = 2$, and $c = \lfloor \frac{97608-88335}{p^{2(2-1)}} \rfloor = 76 \equiv 10 \bmod 11$. For server 3, $j = 3$, and $c = \lfloor \frac{97608-88335}{p^{2(3-1)}} \rfloor \equiv 0 \bmod 11$.

From the above we observe $c \neq 0$ for server 2, which had provided an incorrect share, and we have successfully identified the cheating server.

6 Conclusion

In the proposed improved protocol for Internet voting, the security of the cast ballot depends on numerous servers and the fairness property is satisfied by the use of a ballot partitioning scheme. No encryption/decryption key is used and there is no explicit encryption of the votes. The partitions of the ballot provide implicit security. It overcomes the limitation of lack of redundancy and verifiability in our earlier scheme.

Our proposed method divides the votes into pieces such that loss of some of the pieces (less than threshold) does not result in the loss of the vote. A method for validation/verification of the ballot pieces that are brought together at the time of tally, and, a variant of the protocol is presented that can detect invalid ballot pieces as well as expose the cheating server.

Acknowledgment

This work was partly funded by Center for Telecommunications and Network Security (CTANS), Oklahoma State University, Stillwater, OK.

About the Authors

Abhishek Parakh is with the Computer Science Department at Oklahoma State University in Stillwater. His research interests include cryptography, network security, and signal processing.

Subhash Kak is Professor and Head of Computer Science Department at Oklahoma State University in Stillwater. He has worked in the fields of cryptography, quantum information science, and neural networks.

References

[1] G. Schryen, "Security aspects of internet voting,", *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, pp. 50116b, 2004.

[2] D. Chaum, A. Fiat and M. Naor, "Untraceable electronic cash," *Advances in Cryptology CRYPTO '88*, Lecture Notes In Computer Science, vol. 403. Springer-Verlag, pp. 319-327.

[3] M. J. Radwin, "An untraceable, universally verifiable voting scheme"; available at `http://www.radwin.org/michael/projects/voting.html`

[4] R. Sinnott, Ted Selker, Bil Lewis, Brendan Whelan, James Williams and James McBridem, "Evaluation of voting machine, peripherals and software", *In First Report of the Commission on Electronic Voting on the Secrecy, Accuracy and Testing of the Chosen Electronic Voting System Appendix 2C*, Dublin 2004, pp. 153-191.

[5] G. E. G. Beroggi, "Secure and easy Internet voting," *Computer*, vol. 41, no. 2, pp. 52-56, 2008.

[6] International Herald Tribune. "Estonians will be first to allow Internet votes in national election", Feb 22, 2007. `http://www.iht.com/articles/2007/02/22/business/evote.php` retrieved on March 17, 2009.

[7] S. H. Yun and T. Y. Kim, "Convertible undeniable signature scheme," *Proceedings of IEEE High Performance Computing ASIA '97*, pp. 700-703, 1997.

[8] P. S. DeGregorio, "New voting technology: problem or solution?", *eJournal USA*, October 2007; available at `http://usinfo.state.gov/journals/itdhr/1007/ijde/degregorio.htm`

[9] United States Department of Defense (2007), "Expanding the use of electronic voting technology for UOCAVA citizens", May 2007.

[10] M. Jakobsson, A. Juels and R. Rivest, "Making mixnets robust for electronic voting by randomized partial checking", *USENIX '02*, pp. 339-353, 2002.

[11] D. Chaum, "Secret-ballot receipts: true voter-verifiable elections", *IEEE Security and Privacy*, vol. 2, no. 1, pp. 38-47, 2004.

[12] C. Park, K. Itoh and K. Kurosawa, "Efficient anonymous channel and all/nothing election scheme", *Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, vol. 765, pp. 248-259, 1994.

[13] A. Juels, D. Catalano and M. Jakobsson, "Coercion-resistant electronic elections (extended abstract)", *ACM Workshop on Privacy In The Electronic Society 2005 (WPES '05)*, pp. 61-70, 2005.

[14] S. H. Yun and S. J. Lee, "An electronic voting scheme based on undeniable blind signature scheme", *Proceedings of 37th IEEE Carnahan Conference on Security (ICCST)*, Taiwan, pp. 163-167, 2003.

[15] A. Shamir, "How to share a secret". *Communications of the ACM*, vol. 22, issue 11, pp. 612613, 1979.

[16] Michael Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance", *Journal of the ACM*, vol. 36 pp. 335-348, 1989.

[17] A. Parakh and S. Kak, "How to enhance the security of electronic voting?" *ACM Ubiquity*, vol. 8, issue 6, 2007.

[18] A. Parakh and S. Kak, "A tree based recursive scheme for space efficient secret sharing", *Cryptology ePrint Archive, Report 2009/409*.

[19] A. Parakh and S. Kak, "Space efficient secret sharing: a recursive approach", *Cryptology ePrint Archive, Report 2009/365*.

[20] A. Parakh and S. Kak, "A recursive threshold visual cryptography scheme", *Cryptology ePrint Archive, Report 2008/535*.

[21] M. Gnanaguruparan and S. Kak, "Recursive hiding of secrets in visual cryptography", *Cryptologia 26*, pp. 68-76, 2002.

[22] A. Parakh and S. Kak, "Internet voting protocol based on implicit data security," *Computer Communications and Networks, 2008. ICCCN '08. Proceedings of 17th International Conference on*, pp.1-4, 2008.

[23] T.-C. Wu and T.-S. Wu, "Cheating detection and cheater identification in secret sharing schemes," *Computers and Digital Techniques, IEE Proceedings*, vol.142, no.5, pp.367-369, Sep 1995.

[24] S. M. Sled, "Vertical proximity effects in the California recall election" *VTP Working Pap. 8, Caltech and MIT*, 2003.