Pretty Good Voting

Christian Bell Jason Duell Amir Kamil

December 20, 2004

1 Introduction

Recent academic assessments of the potential for secure Internet-based voting have been extremely pessimistic. A panel assembled to review the Department of Defense's SERVE (Secure Electronic Registration and Voting Experiment) project– a system designed to allow foreign-based members of the armed services to vote online in Federal elections, which was nearly deployed for the 2004 election–concluded not only that the specific design of SERVE was insecure, but that "there really is no good way to build such a voting system without a radical change in the overall architecture of the Internet and the PC, or some unforeseen security breakthrough" [6].

These sorts of pessimistic evaluations have tended to focus on the 'holy grail' of Internet voting: public Federal and State elections. Such elections have a number of properties which make an Internet-based solution extremely difficult. But the fact that online Federal elections are not currently plausible no more invalidates the general concept of Internet voting than the fact that transactions on the New York Stock Exchange are not directly performed on the Internet invalidates the general concept of 'e-commerce'. It is our contention that a wide range of elections are in fact possible to perform with 'good enough' security on the Internet. We believe that such elections will be useful in many contexts, such as elections held by non-profit organizations, corporate shareholder votes, political associations, and other non-governmental entities.

In fact, many such elections are already being held online. Shareholder voting online has become common. Political organizations such as MoveOn.org have held online elections to allow their membership to choose elements of their political strategy (such as which campaign advertisements to run). During the 2004 Democratic primary race, the campaign of Howard Dean held an online election that allowed donors to decide whether his campaign ought to opt out of Federal campaign finance limits. These developments suggest that online voting at a level 'below' that of full-blown public elections is a phenomenon of some social importance. This importance seems likely to increase in the future, especially if holding an online election becomes as convenient as setting up an online auction or discussion board–and there is no inherent reason why this should not be the case.

Of course, the fact that online elections are already being held does not mean they are secure. While we are not privy to the implementations currently used by existing systems, they generally seem to adhere to what could be called an 'e-commerce' level of security: a user sends sensitive information to a trusted website via an HTTPS connection (both to prevent snooping, and to authenticate that the election server is genuine), and the website is then relied upon to do the 'right thing'. In the case of actual e-commerce,

this means that users trust that an online merchant will not disseminate their credit card information, for example. For an online election, it means that the website is trusted with all aspects of running the election: authenticating users, ensuring that only certified voters are allowed to vote (and only once), recording votes faithfully, and performing and publishing an accurate tally of the vote. From a computer security perspective, of course, this is a very problematic model. The single, monolithic election server is a single point of failure, and can be compromised by malicious insiders, buggy code, or outside hackers. Such compromise may be undetectable, as the system is not transparent, and provides no audit trail. Voters in any sort of highly contested election are thus very likely to be wary of trusting such a system, particularly if it is administered by a party that can in any sense be seen to have a stake in the outcome of the election.

At the opposite end of the spectrum, a great deal of recent research has been conducted into secure protocols for electronic voting [2, 1, 9]. However, these protocols often taken for granted a number of assumptions that we see as very problematic for a practical Internet voting system, such as that it is feasible to distribute custom election software to the voting population; that users are capable of generating and managing their own encryption keys; that it is acceptable to use protocols that require votes to be passed from voter to voter to accumulate certain encryption properties; and, most fundamentally, that voters will trust a system which has been blessed by computer scientists and cryptographers, despite being opaque and unintuitive to laypeople.

In this paper we try to find a middle ground between these approaches. Clearly, a more secure model than the 'trusted website' needs to found. But on the other hand, we are interested in a design that is practical to deploy on the Internet today, given current Internet technologies and current personal computing environments. We are also committed to a system that we feel is likely to be trusted by 'naive' users who know little about encryption (or other computer security issues), and who may be capable of little more than operating a browser and an email client. (i.e. no key generation or management). We believe that this constrained design space still offers enough room to construct Internet voting systems that provide acceptable guarantees for the election security parameters that voters care most about (first and foremost, that election results are fair and accurate; secondly, that voter privacy is maintained), while punting on other issues (most notably, vote selling and coercion) that we feel in practice are likely to be detectable and remediable by social processes, and so can remain outside the province of election software.

We present two versions of a 'pretty good voting' (PGV) system. While both offer similar, strong guarantees of election fairness, they differ in the degree to which they make it difficult to trace votes back to individual voters (and inversely, in how difficult they are to implement using a standard web browser as the voting client). One version provides a 'weak' guarantee of voter anonymity which is essentially analogous to e-commerce sites' protection of credit card data. The other provides a stronger privacy guarantee, requiring collusion between many components to connect voters to their ballots, but at the cost of requiring non-standard (i.e. not just SSL) encryption to be performed inside the user's browser, which is likely to be less portable across the disparate browser codebase present on users' machines today.

Two key ideas motivate both of our designs. First, we emphasize the need to not trust any one element of a voting system, and that when several elements need to be trusted in tandem, it is best to run them until the control of each of the contesting parties in the election. This is analogous to the traditional polling place model, in which representatives from each party guard the ballot box in order to detect foul

play. Second, our voting system relies heaviliy on receipts. This is in sharp distinction to most other voting systems, which seek to be receipt-free (in order to prevent coercion, which can occur if a voter has a method of demonstrating how they voted after the fact). We feel that the ability of our system to allow voters to locate their ballots in plain text in the election results is vital to voter trust in the accuracy of results, and this property also provides an extreme level of transparency that makes it very difficult for any components of our system to tamper with the election results without being detected.

2 Security Goals

There are a number of well-known properties that are desirable for any election (whether run on the Internet or not). There are also a number of well-known security problems with the Internet, and with software running on general purpose machines. In this section we outline what the security goals for our PGV system are with respect to these issues.

2.1 Election Requirements

- *Fair elections*: The most single fundamental requirement of any election is that it be fair, i.e., only registered voters may vote (and vote only once), and the outcome of the election must be an accurate tally of these voters' votes. We believe that mistrust of Internet voting is largely based on the fear that this will not be guaranteed, and that votes can be 'rigged'. We believe that both of our PGV designs produce strong guarantees for fair elections.

- *Fraud detection and recovery*: If any party tampers with the fairness of an election (intentionally, or inadvertently), this must be detected. Ideally, this can be done in a way that allows the election to proceed. PGV puts an emphasis on detecting tampering, and on identifying the entity that is causing the tampering. However, no particular policy is built into the system with respect to recovery, as we see the 'right' approach as often being a matter of policy. For instance, if it is detected that one out of a number of servers is reporting different results from the rest, it may acceptable to discount results from that server, and continue an election. The tolerable ratio of misbehaving entities to behaving ones is up to the election participants (who should obviously decide this in advance of the election). Given the high likelihood of detection, however, we feel fraud is unlikely to occur frequently. Coupled with the fact that Internet elections are easy to start over, this means that the appropriate response to fraud detection is to halt the election, deal with the suspicious parties in a social manner (be it taking legal action, or simply booting them out of the process), and then re-running the election.

- *Anonymity*: Ballots should not be tracable back to the voters who cast them. This is a loss of privacy, and can also open the door to coercion and/or vote selling. While we believe that anonymity is not as essential as fairness for the types of elections PGV is targeted to. Voters in many contexts are much more concerned that an election be held fairly than they are that their particular vote may be knowable to others. They may thus be willing to accept lesser guarantees about their anonymity. That said, string anonymity guarantees are obviously worth pursuing whenever possible.

The 'weak anonymity' version of PGV essentially provides e-commerce grade protection for anonymity: votes are only exposed in a way that clearly ties them to voters within trusted servers. So long as these trusted servers behave, no one else will be able to correlate voters with their votes. On the other hand, if anonymity is compromised, there may be no easy way to trace which party is responsible. This is exactly analogous to the way that users trust multiple e-commerce sites with personal information like their credit card number, address, and purchase history. This has proven acceptable to users in many Internet contexts, so long as a website has a trusted record with respect to maintaining user privacy, and we see no reason why it cannot be the same for a large set of election contexts as well.

The 'strong anonymity' version of PGV provides much stronger guarantees. No one element of the system is able to tie a voter to a vote, and so long as one element in a chain of mix net servers is 'honest', none of them will be able to gather this information.

- *Receipt-freeness*: Traditional voting models consider the ability of a voter to provably find their ballot in election results to be a liability, even in a manner that does not automatically compromise their identity (i.e., a vote can be found not via a voters' name, but by a ballot number, or a 'magic phrase' that they pick), since this opens the door to both coercion and vote selling. We agree that these behaviors become possible once a receipt is provided, but maintain that the benefits of providing receipts outweigh these factors in a large set of electoral contexts. Receipts provide an enormous amount of transparency, making it extremely hard to meddle with election results. We also believe they will provide a high level of trust in the fairness of election results. Attempts to coerce and/or buy votes, on the other hand, are realistic threats only in a small subset of electoral contexts. In most elections, spending to advertise one's positions is a better use of one's money than is an outright attempt to purchase votes. Attempts to purchase votes are likely to be made to parties that find them offensive, resulting in a strong possibility of detection and punitive measures. Similarly, coercion requires the threat of coercion to be viable and strong, but in many contexts detection and punishment is a more likely outcome. Finally, both coercion and vote selling are "one voter at a time" methods of tampering with election results, and require more effort to get more results, whereas tampering with software presents a much greater danger of radically (and undetectably) altering elections. We thus believe that a large class of Internet elections will benefit much more from the increased tamper-resistance that receipts offer than they lose by not having them.

2.2 Computer Security Goals

- *Network security*: A basic issue for any Internet voting system is how to ensure that parties are able to authenticate themselves to each other, and how to secure their communication from other parties. PGV relies on the existing architecture of DNS and HTTP/SSL (HTTPS) to provide these guarantees. In order to masquerade as a server in the election system, one would need to compromise both DNS, and obtain the real server's SSL key. While this is by no means impossible, it is difficult to do on a large scale. Even if it is accomplished, and a malicious server can transparently place itself between voters and the real servers in the election system, the receipt-based nature of our system makes it difficult to tamper with an election without being detected.

- *Availability*: Network availability is one of the hardest issues to guarantee in the current Internet environemnt, which offers no proven way to prevent denial of service attacks. In practice, however, denial of service attacks have so far always been temporary. PGV assumes that elections can be occasionally delayed by DoS attacks.

- *Malicious client software*: Another difficult issue for any Internet voting system is the possibility that voters' browsers (or other software, potentially even including their operating system) may be compromised, and may thus not behave as the user wishes it to (even while deceiving the user into thinking that it is). PGV has two lines of defenses against such attacks. First, the receipt-based nature of the system means that malicious code needs to not only miscast votes, but must also be able to deceive users when they check their ballots. By providing a variety of mechanisms for viewing their ballot (including the ability to use a browser on any other machine to check their vote), the task of constructing a 'seamlessly' deceptive user experience becomes exponentially difficult. Secondly, we have constructed a set of optional additions to our system which use out of band information (sent by physical mail or other trusted means) to defeat malicious client code.

2.3 Related Work: Traditional Voting and e-Voting

Although individually listing some of the properties required in an e-voting system and explaining some of their implications explores interesting facets of the problem space, additional difficulties surface when comparing e-voting systems to traditional voting systems. Here are some points to consider when comparing both types of systems, as well as notes on how PGV would sit alongside both traditional and existing e-voting approaches [3, 4].

Single points of trust. Traditional election systems carry out the notion that no single person can be entrusted to participate solely in any of the basic election steps. Mutually mistrusting parties are encouraged to take part in validation, collection and tallying. The fact that elections are decentralized and that this decentralization is supported as a model for fair elections limits the impact of large-scale fraud. Existing voting systems, including PGV, tend to suggest that holistic approaches are preferred, mostly because of the difficulties in securely delegating electronic trust and for simpler reasons such as claiming that the proposed systems have figured out the "right" way to approach e-voting. Existing deployable e-voting protocols [4] decouple the administration of the vote which validates the right for one to vote from the actual casting of the vote. In this case, it is assumed that the administration is to be trusted as a single point of trust and that the ballot casting and that the vote collection is to be done through one or many parties. In PGV, we stress the fact that in any areas where single points of trust are technically feasible, any steps taken by the single party in question is validated by a set of observers (more in section 3).

Audit trails. Traditional elections are powerful in the sense that the entire election may be replayed, or recounted. However, audit trails have recently been the subject of controversy in the wake of certain DRE machine failures, where a single computing platform is used to store vote counts without providing any audit trail in paper form [7]. E-voting typically follows in DRE guarantees as to allowing elections to be easily replayed. Although audit trails are important in elections where the stakes are very high and where restarting an election is not an easy task, systems such as PGV do not consider paper audit trails to be a de facto property of the election protocol. Rather, the election can be audited through the use of

independent observers that actively verify the election and provides means for external users see actions taken by entities collection, posting and tallying the votes.

Physical security and integrity. Physical security lies at the very difference between e-voting and traditional systems, where orderly conduct and voting protocols may be applied to registered and potentially non-registered voters. If need be, security personnel can make sure that mischiefs cannot disrupt the electoral process, which in contrast, in not possible with e-voting due to the threats of global-scale attacks in the form of denial of service. Also, physically depositing a ballot shows that a user's vote has reached it's intended destination (ballot box) and does not suffer from communication failures which may prevent a voter from truly knowing if her/his vote was cast. With PGV, we actually believe that there is some benefit in allowing the users to see their vote for exactly these reasons.

3 PGV Architecture

Figures 1 and 2 show the two variants of our PGV voting system. In each, responsiblity for authenticating users and gathering results is split between a set of authentication servers, which authenticate voters, and observers, which monitor that the various authentication servers produce the same results, which the observers then keep as a matter of public record. In the 'hard anonymity' version of PGV, a mix net is also used to decouple voters from their ballots.

The two protocols share some common elements:

- **Authentication Servers**. A set of three (or more) *authentication servers* is authorized to receive votes from users. As an analogy to traditional voting, these servers should be run by different organizations with opposing objectives in the election and preferably with different code bases. Each organization's authentication server may be mirrored to provide higher availability and to defend against DOS attacks, but care must be taken to avoid other attacks that take advantage of this duplication (§5.3).

  Each authentication server is provided with a list of all registered voters and one-way hashes of their passwords, along with any other user-specific information required by the election procedure. Servers only accept votes from registered users.

  Authentication servers report their votes directly to a set of observers. Votes are assigned sequence numbers and signed by authentication servers in order to prevent observers from inserting fake votes. In order to prevent observers from deleting votes, each authentication server also signs a statement at the end of the election that contains a cryptographic hash of all votes received and a number denoting how many total votes there were.

- **Observers**. A larger set of *observers* receives votes from authentication servers and monitors their behavior for cheating. Observers compare votes receieved from the different organizations and make sure that the sets of reported votes do not differ. This can be done in real time with only very loose synchronization by allowing votes received in the previous N amount of time to differ, N potentially on the order of minutes. An observer can guarantee that it has all of a server's votes so far using the sequence numbers assigned by that server.

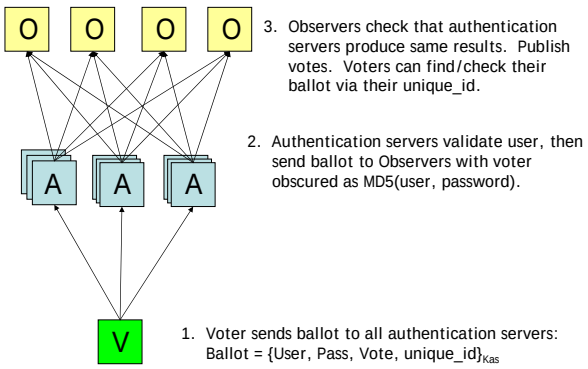

Figure 1: Overview of the PGV Weak Protocol

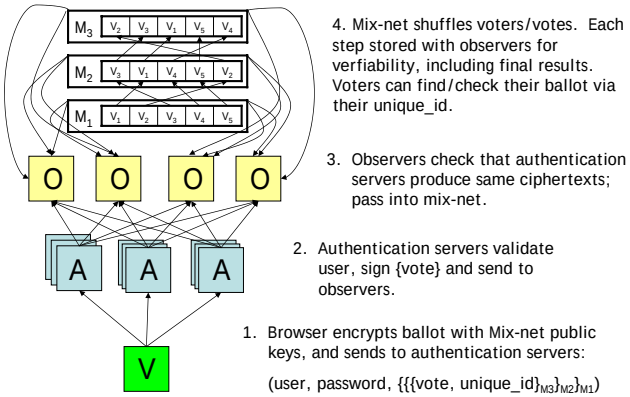

Figure 2: Overview of the PGV Hard Protocol

Observers also act as a *bulletin board* where other authorities in the voting process can retrieve and send information. The observers also store the final list of decrypted votes, enabling users to query them directly in order to check their votes or the overall results.

- **Tallying**. Once the observers have the final list of votes, voting authorities can obtain it from the observers in order to tally the results. Other interested parties can also obtain the list in order to verify the official tallies.

- **User Verification**. The Pretty Good Voting protocol allows users to verify that their ballots were cast as intended. Voters are allowed to embed a voter-defined string in their ballots that gets encrypted along with the rest of the ballot. When the final set of votes is published, each vote will appear with its corresponding voter-defined string. Users can verify the correctness of their votes by searching for their strings and confirming that the corresponding votes are as they intended.

  This verification procedure is not without flaws. Users are encouraged to choose a value for this field which is likely to be unique among all ballots cast, but no enforcement of this is done (in order to avoid synchronization and coordination issues among the servers). Also, voters can abuse the verification mechanism in order to cast doubt on the election. Section 5.3 discusses this issue in more detail.

The diagrams here only display the path that a voter's ballot takes through the system. To get to this point, the voter will have gone to the organization which is holding the election, and received an HTML page which contains a ballot form, which, when submitted, casts to each of the authentication servers the voter's choices, their user/password (unique passwords for each election are likely to be desirable, but this is not required, and is ultimately a policy choice), and their user-defined string.

3.1 Weak Protocol

In the weak anonymity protocol, clients send their votes to authentication server with no additional encryption besides SSL. Authentication servers, after verifying that the client is registered and hasn't yet

voted, apply an MD5 hash to the client's username and password and transmit the hash and vote to the observers.

Once the observers have received all of each authentication server's votes, they can proceed to publish them to the general public. Each ballot is published together with its corresponding username and password hash and user-defined string, as in table 1. An advanced user can directly obtain his vote by applying the MD5 algorithm to his username and password and searching for the resulting hash. Less proficient voters can use their voter-defined strings instead to search for their ballots.

Since authentication servers receive ballots unencrypted from clients, each server knows how each voter cast his vote. Thus, each individual server has the ability to reveal a voter's identity. This is analagous to security in e-Commerce, where each website at which a user has purchased anything can reveal the user's credit card information.

Total: 123,456, Hash: c09a7347eb1d2...			
SeqNum	MD5(User, Pass)	User String	Ballot
1	a3dc767be7b9c...	my pretty pony	[Pres: J. Smith, ...]
2	7a8cdf7b17e23...	3.141592653589	[Pres: J. Schmoe, ...]
⋮	⋮	⋮	⋮

Table 1: Output of the authentications servers in the PGV Weak Protocol. Signatures have been omitted. This is also the final published vote list.

3.2 Hard Protocol

Stronger anonymity guarantees can be provided by modifying the PGV protocol to use *mix nets* [5]. A mix network consists of a set of servers each of which permutes the input set in turn, so that tracing an output back to its input requires the cooperation of all servers in the set. Two types of mix nets can be used, *decryption* mixnets and *re-encryption* mix nets, each requiring a different PGV protocol:

- **Decryption Mix Nets**. In a decryption mix net scheme, users encrypt their ballots in turn using each mix server's public key. The result resembles an onion, with each onion layer corresponding to a layer of encryption. Mix servers peel off their layers of the onion by decrypting with their private keys and proceed to permute the ballots before sending them to the observers to be used by the next mix server.

Total: 123,456, Hash: e65cf56b5ae6c...		
SeqNum	User	Ballot
1	jduell	$\{$my pretty pony, Pres: J. Smith, ...$\}_{K_m}$
2	kamil	$\{$3.141592653589, Pres: J. Schmoe, ...$\}_{K_m}$
⋮	⋮	⋮

Table 2: Output of the authentications servers in the PGV Hard Protocol using a re-encryption mix net. Signatures have been omitted.

Total: 123,456, Hash: 12b2c8a81d4ea...	
User String	Ballot
my pretty pony	[Pres: J. Smith, ...]
3.141592653589	[Pres: J. Schmoe, ...]
⋮	⋮

Table 3: Final published vote list in the PGV Hard Protocol. Signatures have been omitted.

In order to prevent adversaries from just re-encrypting a mix server's outputs with its public key to determine the corresponding inputs, users should augment their ballots with a nonce before applying each key. Mix servers should then strip the nonce when decrypting.

- **Re-Encryption Mix Nets**. Re-encryption mix nets rely on encryption protocols such as ElGamal that allow cyphertexts to be re-encrypted with knowledge of only the public key, without affecting the ability of the cyphertext to be decrypted. Users need only encrypt their ballots a single time with the mix net's public key. Mix servers re-encrypt the mix network's inputs in turn, permute them, and send observers to be obtained by the next server.

  In order to prevent the first server in the mix net from revealing users' identities, re-encryption mix nets typically decouple mixing from actual decryption of the inputs [2]. The private key is shared among a set of *decryption servers* using some sort of secret sharing scheme, whose parameters can be set such that at least a majority of decryption servers is required to compromise anonymity. The decryption servers collectively decrypt the ballots once they have been permuted by the mix servers.

In either case, clients send their encrypted ballots to the authentication servers, which in turn sign and send each client's username and ballot to the observers. Observers publish these (username, ballot) pairs as in table 2, allowing advanced users to check to make sure that the cyphertexts they sent to the encryption servers are the same as those recorded by the observers. The first mix server then obtains the signed ballots from an observer, using the signatures to verifying that the inputs are complete and accurate. The server strips off the username and proceeds to mix the votes, signing the results and sending them to the observers. Each proceeding mix server then obtains its inputs from the observers, repeating the process.

In the decryption mix net scheme, the output from the last mix server is the final list of unencrypted ballots and can be published as before. In the re-encryption scheme, the output is still encrypted, and the decryption servers obtain it from the observers, jointly decrypt the ballots, and send the results to the observers to be published as in table 3.

Techniques exist for both decryption and re-encryption mix nets to verify that the outputs of each mix server and the decryption servers are equivalent to their inputs. These range from zero-knowledge proofs [9] to randomized partial revelation of permutations [5, 2]. Thus no trust in the mix net is required for correctness.

4 PGV Setup Procedures

Before conducting an election using Pretty Good Voting, a voting authority (VA) must delegate the various roles in the election and provide all parties with their required information. This includes determining who the various authentication servers, observers, and if required, mix and decryption servers are. The VA oversees any required generation and sharing of encryption keys and obtains public keys from all authorities. Finally, the VA writes the client voting software using Java applets, JavaScript, or some other widely available browser plugin, integrates the necessary keys and server addresses into the program, and chooses an SSL-secured website to run the election on.

Voters choose usernames and passwords and register with the VA using some out-of-band mechanism such as *snail mail*. The VA generates any required shared secrets between voters and the authentication servers. It then distributes usernames, passwords, secrets, and the election website URL to voters, and usernames, password hashes, and secrets to the authentication servers, again using an out-of-band mechanism.

It is assumed that the above steps are done under the observation of mutually distrusting parties, so as not to allow a single authority to manipulate the election. Cheating in the setup procedure is out of the scope of detection and prevention by PGV.

5 Threat Model

Pretty Good Voting is subject to attacks from various entities, with varying degrees of effectiveness. In this section, we discuss possible attacks against the system, and how such attacks can be defended against.

5.1 Attacks By Authorities

The authentication servers in the election can try to tamper with the election in various ways:

- **Modifying Votes**. Authentication servers could modify users' ballot before posting them to the observers.

  Defenses. Since the observers monitor whether the authentication servers produce the same results, this will be detected as an anomaly unless all authentication servers collude. Under the PGV 'strong anonymity' model, the voter's browser has also encrypted their vote, and so the mix net will detect the tampering. In both models, the voter will be able to check their vote and see that it has been tampered with, so detection is extremely likely even under the 'weak' model. Since authentication servers sign the information they pass on to observers, there will be a trail pointing to the malicious authentication server.

- **Injecting Bogus Votes**. Authentication servers could inject false votes into the system.

  Defenses. Any attempt to inject false votes into the system requires all authentication servers to collude, else the anomaly will be detected by the observers.

  If full collusion is present, votes cast with bogus voter IDs will still be detected by the organization that is holding the election (and/or the observers, if they have been provided with a list of valid voters during election setup).

  Votes cast on behalf of registered voters who did not vote present more of an opportunity.

  An email can be sent to everyone who votes, making it likely that some nonvoters will cry foul if they get an email stating that they voted. Of course, this email can't be sent by any of the authentication servers, else they could simply not send the email for voters that they're "voting for". So an observer must be the one sending a user email. One observer can be elected to do this (in which case you'd need all the authentication servers, plus that observer, to cast votes on

behalf of nonvoters). Or multiple observers could each send an email (all of them would have to collude, but the inconvenience would be that users would get multiple emails). Or a method can be implemented that makes it hard to know in advance which observer will be responsible for sending the email to a particular user. If, for instance, the observers take the final re-encrypted ballot blob from the mix net, and use it as the input to a hashtable that reveals which observer will send the email for that user, then presumably it ought to be hard to predict (from the colluding authentication servers' perspective) which observer will wind up being called on to email the voter, making it difficult to "drop" the emails for certain voters, even if some number of the emailing observers are also colluding. Avoiding detection for any significant number of fake votes in this case would require all the observers to collude, as well as all the authentication servers.

- **Compromising Voter Privacy**. Authentication servers could reveal which voters cast which ballots.

  Defenses. In the 'weak anonymity' PGV model, this is possible, and is explicitly acknowledged in the design. The main defense is the social repercussion if an organization's authentication server is caught doing this.

  In the 'strong anonymity' PGV model, authentication servers never see the user's unencrypted ballot, and the mix net prevents them from correlating the users with the final results of the election, so this attack is not possible.

5.2 Attacks By Mix Net Servers

- **Tampering With Ballots**. Mix nets could attempt to produce bogus plain text ballots from the encrypted inputs.

  Defenses. The encryption algorithms used by mix nets provide auditable checks to see if ballots have been tampered with, and which mix net server did so.

- **Compromising Voter Privacy**. Mix net servers could reveal which voters cast which ballots.

  Defenses. The design of mix nets guarantees that even if only one mix net server is 'honest', the others will not have enough information to correlate voters with their plain-text votes. This attack is thus possible, but only if all the mix net servers collude together to do so.

- **Refusal to Decrypt Ballot**. In a de-encryption mix net, one or more servers could refuse to participate in the mix net process. Since each layer of encryption needs to be removed before passing the ballot on to the next mix net server, this could comprise a denial of service attack.

  Defenses. The mix net servers could be required to use 'shared secret' keys, in which case some combination of the other mix net servers would be able to perform the decryption if a particular server decides not to participate. Obviously, the number of mix net servers that could drop out before the election was blocked would depend on how many share their keys. On the downside, the presence of shared keys could allow a subset of mix net servers to compromise voter privacy (rather than requiring all of them to collude).

5.3 Attacks by Voters

The inputs to the election are provided by voters, so they can tamper with the voting system in an attempt to subvert it. In addition, the design of PGV involves voters in the validation process, allowing them to verify their votes, thus opening up more avenues of attack. Voters can mount the following attacks:

- **Multiple Voting**. Voters can cast multiple, potentially conflicting ballots for the same election. In the case of mirrored authentication servers, votes can be directed to different mirrors of each server.

  Defenses. When an authentication server records a vote from a user, it removes the user from its list of eligible voters, preventing the user from voting on that server again.

  In the case of mirroring, the PGV protocol must be modified to take into account multiple conflicting votes from the same user. Some fair, deterministic choice must be made, such as requiring the earliest or latest vote to stand.

- **Selective Sending of Votes**. A voter can arrange to send a vote to some authentication servers but not others, or to send different votes to different servers, in an attempt to acccuse servers who do not receive the vote of cheating.

  Defenses. This particular defense is difficult to defend against. The majority result can be taken, or if none exist, the vote thrown out entirely. Alternatively, the set of authentication servers can be modified to use a *Byzantine agreement* [8], allowing for a higher-level of tolerance of such events.

- **False Claims of Fraud**. Since voters receive anonymous receipts of their votes in order to detect vote fraud, they can claim that their vote was not recorded as cast, even if it was.

  Defenses. Claims of fraud should be investigated thoroughly by the voting authority, requiring claimants to reveal all secret information. The claimant's vote should then be traced through the system, potentially requiring a mix net to reveal its permutation of that particular vote, until the source of discrepancy, if any, is determined, and to verify that the allegedly fraudulent vote actually was cast by the voter.

- **Vote Transfer**. Voters can transfer their votes to others, whether for profit or due to coercion, using the verification system to prove that they voted in a particular way.

  Defenses. Vote transfer can be defended against as in traditional voting, using social and legal means. Large-scale transfer is very likely to be detected and can subsequently be dealt with through an out-of-band mechanism.

Since voters can only affect their own votes by cheating, and only in such a way as to subvert their intended votes, it is likely that such cheating will only occur on a small scale. It is therefore unlikely to threaten the outcome of an election and should not prevent results from being validated in such a case. In a case where the outcome could be affected, individual cases of cheating should be examined thoroughly and dealt with in a predetermined way, such as voiding all possible cheating votes.

5.4 Attacks by Third Parties

A range of possible attacks can be perpetrated by third parties not involved in the election. Besides malware, discussed in §5.5, the following attacks are possible:

- **Denial of Service**. Adversaries can mount denial of service (DOS) attacks against authentication servers, preventing users from voting. Attacks on observers and mix servers are also possible, but serve only to delay reporting of results and don't affect the actual results of the election.

  Defenses. Authentication servers mirrored in order to provide higher availability in the face of a DOS attack. However, this opens up the door for other attacks, such as multiple voting (§5.3). It also complicates reporting of votes to observers, requiring each mirror to have its own set of sequence numbers, to attach its ID to each reported vote, and to report the total number of votes it recorded.

  An alternative solution is to extend the election window to allow blocked users to recast their votes. This solution would be acceptable in organizational elections, where a strict election date and time isn't required.

  Unfortunately, the DOS vulnerability appears to be fundamental to the nature of the Internet [6]. There is no current solution to completely defend against such attacks.

- **Spoofing of Authentication Servers**. An adversary can masquerade as an authentication server to voters, using flaws in TCP, DNS, or other protocols. The adversary can then selectively discard votes that it receives.

  Defenses. SSL encryption prevents most spoofing. Combined attacks on DNS and SSL are avoided by sending voters the actual SSL secured URL of the election website (§4) and forcing them to directly connect to it.

  PGV also defends against spoofing by using secrets shared between voters and authentication servers (§5.5). Voters can detect spoofing by comparing the supposed secrets received from the authentication servers with those on their paper ballots.

- **Masquerading as Observers**. Adversaries can pretend to be election observers and provide false information to other election authorities or to endusers.

  Defenses. In PGV, any election authority that posts information to observers signs it first, and any authority that receives information checks the signatures to make sure it wasn't tampered with. Fake observers are thus no more than a nuisance to the election process.

5.5 Malicious Code

The first step in dealing with malicious code is to realize that it is an inescapable threat if any type of voting is to be carried out over commodity user platforms (operating systems and network environment). Since PGV assumes that these commodity platforms are used to vote and that the claim is that it is still possible to provide voting protocols in their presence, malicious code must be thwarted in some way.

Given that election results and individual votes are publicly posted at the end of the election, the presence of malicious code can be detected by voters if they see that their vote is not appropriately

represented in the tally. We classify this approach of detecting malicious code through voter receipts as *late* detection. Since the current version of PGV does not provide any mechanism for voters to recast their ballots, the only obvious solution is to restart the election unless malicious code can be detected earlier.

The PGV approach to minimizing the effect of malicious code on election outcomes uses an out-of-band mechanism such as *snail mail* to propagate information that the user should expect at voting time. The out-of-band communication between the voting authority (VA) and the voter (V) works as follows and provides *early* detection of malicious code:

- VA selects N meaningful images for the c possible choices on the ballot where $N \gg c$ (example: A list of different types of fruit).
- VA generates all possible k subsets of c images where $k = \frac{N!}{(N-c)!}$ which establishes k different ways to create a ballot with c choices given N possible images to represent each choice.
- For each voter, VA generates a set of random confirmation strings, one for each authentication server.
- VA assigns, at random, one of the k image subsets k_i to each voter, and uses *snail mail* to notify the voter of this subset and the voter's confirmation strings.
- Upon receiving the subset assigned to him/her, V knows what that subset maps to in terms of actual choices on the ballot.
- At election time, each authentication server presents V with what is supposed to be the subset that was assigned to V.
- V makes sure that what each authentication server shows on the voting page corresponds *exactly* to the images shown on the paper ballot received through the mail.
- V votes for the image of its choice.
- Each authentication server replies with its confirmation string for V. Only if all confirmation strings match those on V's paper ballot can V be sure that the vote was recorded.

Malicious code can either act with the intent to balance an election in a certain direction or simply to wreak havoc. The scheme described above prevents malicious code from acting for profit, since the voter and the voter's interface share a secret that can only be guessed at by the malware.

However, the scheme does not address the case for which an ill-intending party having control of a voter's browser wishes to send anything *but* the voter's actual ballot selection, having determined, for example, that the voter is a member of an opposing party. This, in many ways, ruins the efforts of providing early detection. For this case, we have imagined a scenario where the voting authority generates subsets with c much larger than the actual amount of choices available on the ballot. This approach would require that voters spend more time comparing their on-screen voting forms to the mail ballots but in the "simple" version of the PGV protocol, has the advantage of allowing authentication

servers to detect bogus votes early and warn the user that they have either miscast their vote or that their voting platform is contaminated.

One last aspect of this effort to mitigate the effects of malware is the fact that it cannot easily support write-ins. In the presence of text areas available to the user to send in write-in votes, malicious codes could simply elect to always use the write-in capabilities to send in bogus values and trick the user into thinking that their selected image has been sent in as a vote. An alternative would be to have every voter send a write-in whether the intention is to do so or not and have the actual write-in voters couple their write-ins with a special image dedicated for write-in votes. Allowing these types of votes does have the drawback of providing users with an easy way out of the malware-avoidance puzzle we are proposing since it is simpler to type in a write-in vote than to actually spend the little time necessary to associate a ballot selection to a paper image. For this reason, and as a general principle, we consider write-in capabilities to be difficult to provide in the presence of heuristics to detect malicious code.

6 Conclusion

We believe that the PGV system described in this paper represents a contribution to the possibility of secure yet practical Internet voting systems. Our system provides very strong guarantees that elections are either fair, or detected as fraudulent/flawed. It provides a very intuitive model for users to understand, and a receipt-based method for them to check election results that we believe would instill confidence in the system among voters. The lack of such trust is one of the primary obstacle to online elections becoming acceptable, and so we assert that the trade-offs to having receipts (coercion, vote-selling) are worth it, at least in a large enough set of election contexts. Our system allows for the use of standard Internet browser technology, while providing plausible defenses against compromised client systems interfering with elections. Overall, we have not identified any way of attacking our systems that allows any party to tamper significantly with the fairness of an election without detection. While the ideal goal of providing online governmental elections is not forwarded by our work, we believe that the subset of election than can be covered by PGV is interesting enough to merit our work.

References

[1] A. Acquisti. Receipt-free homomorphic elections and write-in ballots. Cryptology ePrint Archive, Report 2004/105, 2004. http://eprint.iacr.org/.

[2] D. Boneh and P. Golle. Almost entirely correct mixing with applications to voting. In *Proccedings of the 9th ACM Conference on Computer and Communications Security*, November 2002.

[3] L. Cranor. Sensus. http://lorrie.cranor.org/voting/sensus.

[4] A. F. et al. A practical secret voting scheme for large scale elections. *Lecture Notes in Computer Science*, 718:244–251.

[5] M. Jakobsson, A. Juels, and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking.

[6] D. Jefferson, A. Rubin, B. Simons, and D. Wagner. A security analysis of the secure electronic registration and voting experiment. Technical report, Department of Defense Federal Voting Assistance Program, January 2004. http://www.servesecurityreport.org/.

[7] M. McGaley and D. J. P. Gibson. E-Voting: A Safety Critical System. Technical Report NUIM-CS-TR-2003-02, NUI Maynooth, Computer Science Department, 2003. http://www.cs.may.ie/research/reports/2003/index.html.

[8] M. K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in rampart. pages 68–80.

[9] K. Sako and J. Killian. Receipt-free mix-type voting schemes – a practical solution to the implementation of a voting booth. In *Advances in Cryptology – EUROCRYPT '95*, pages 393–403, 1995.