Remotegrity: Design and Use of an End-to-End Verifiable Remote Voting System*

Filip Zagórski[1], Richard T. Carback[2], David Chaum[3], Jeremy Clark[4], Aleksander Essex[5], and Poorvi L. Vora[6]

[1] Wrocław University of Technology
[2] Draper Laboratory
[3] Voting System Institute
[4] Carleton University
[5] Western University
[6] The George Washington University

Abstract. We propose and implement a cryptographically end-to-end verifiable (E2E) remote voting system for absentee voters and report on its deployment in a binding municipal election in Takoma Park, Maryland. Remotegrity is a hybrid mail/internet extension to the Scantegrity in-person voting system, enabling secure, electronic return of vote-by-mail ballots. It provides voters with the ability to detect unauthorized modifications to their cast ballots made by either malicious client software, or a corrupt election authority—two threats not previously studied in combination. Not only can the voter detect such changes, they can prove it to a third party without giving up ballot secrecy.

1 Introductory Remarks

In 2009, the city of Takoma Park in Maryland, United States, became the first election authority (EA) to use a cryptographically end-to-end verifiable (E2E) voting system in a public election [4]. This system, Scantegrity II [7], allows voters to verify their votes were counted correctly, while maintaining ballot secrecy. Scantegrity also provides a *dispute resolution* mechanism: in the event either the voter or the EA behaves maliciously, parties that follow the protocol should be able to prove their honesty to a third party (such as a democracy watch group). These integrity and dispute resolution protections afforded by the in-person nature of Scantegrity II, however, do not immediately extend to absentee voters submitting ballots by mail or online.

Shifting from in-person to remote voting introduces new threats, including the possibility of malicious software on the voter's computer making unauthorized (and potentially undetected) modifications to ballot selections. Although

* Full version available: `http://eprint.iacr.org/2013/214`. Zagórski was funded in part by NSF Awards 0937267 and 1137973 and by the Polish National Science Center (NCN) scientific project 2010-2013 with grant number N N206 369839. Clark and Essex acknowledge funding through NSERC PDF awards.

M. Jacobson et al. (Eds.): ACNS 2013, LNCS 7954, pp. 441–457, 2013.

this threat has been well studied in isolation, a major complication arises when simultaneously considering the problem of dispute resolution: a malicious EA caught cheating could spuriously blame the voters' clients for the malfeasance.

In this paper we tackle the problem of protecting against malicious software on the voter's computer while simultaneously offering a dispute resolution procedure. To that end we present Remotegrity, a remote voting extension for Scantegrity designed to extend similar protections to absentee voters as those of voters attending the polling place. We propose the Remotegrity protocol and describe an implementation which was fielded in Takoma Park's municipal election in November 2011.

Contributions. The main contributions of this paper include:

1. The Remotegrity protocol, a remote voting system providing voters with the ability to detect and prove unauthorized changes made to their ballots by malicious client software or a corrupt election authority,
2. An implementation and case study of Remotegrity in a municipal election,
3. Lessons learned from the real-world deployment of voting systems research.

2 Background

Absentee Voting. A reality of elections is that a certain portion of the electorate will be unable to physically attend a polling place during the election period, *e.g.,* due to illness, travel, or residing out of the district. Four common methods for enfranchising absentees exist. *Early voting* is most appropriate for travellers but does not assist the ill or non-resident. *Vote-by-proxy* breaches ballot secrecy and is not generally used in public-sector elections. Hosting a *polling place abroad* is suitable when a large contingency of absentees are local to the area, such as a military base or embassy in a large foreign city. It is less suitable for small-scale, *e.g.,* municipal-level, elections.

Most EAs use both early voting and a fourth method: *remote voting*. Remote voting could be either (i) available only to voters demonstrating a need, (ii) available to any voter, or (iii) mandatory for all voters. In the United States, there are respectively 27, 21, and 2 states/capital districts in these categories at the time of writing.[1] In addition 33 offer early voting.

The primary method for delivering and receiving ballots from remote voters in the United States is the postal system. Vote-by-mail enables threats not present in polling place voting: ballots could be mailed to the wrong address or lost before being received by voters; voters can demonstrate how they vote for payment or be coerced into voting a certain way; there may not be a strong mechanism to authenticate that a ballot was filled out by the intended voter (or distinguish a real ballot from impersonated fake ballots); ballots could be lost, delayed, or tampered with during their return to EA; and there are only weak guarantees of ballot secrecy from the election officials receiving the ballots.

[1] Absentee and Early Voting. *National Conference of State Legislators*, 4 Sept 2012.

Online Voting. Of the issues with vote-by-mail, the most significant is arguably that ballots are not always received in time—19% of mail-in ballots cast in the 2008 US election were not received in time to be counted. In response, election officials are interested in enabling electronic channels, such as email, fax, or the internet for voters to receive and return ballots. In addition to subsuming most of the issues with postal ballots, online voting introduces several of its own. Malware on a voter's computer may undetectably alter the voter's choices. Email and fax do not provide secure transport for ballots, and while websites can, this requires the assumption that voters can correctly authenticate the server (*e.g.,* voters do not fall prey to phishing, SSL-stripping, or man-in-the-middle attacks with illegitimately obtained certificates [10]). The EA servers may be made inaccessible through a denial-of-service attack. Most importantly, a compromise of the server could allow all cast ballots to be undetectably modified.

Hybrid Internet/Mail Voting. The delay introduced by the postal system can be partially addressed by utilizing an electronic channel only for ballot receipt, or ballot return. In many U.S. counties and states, blank ballots can be downloaded and submitted by mail.[2] Conversely, ballots are received by mail and submitted online in Remotegrity. Given that the date a voter receives a blank ballot is a soft deadline, whereas the date the EA must receive the returned ballot is a hard deadline, it is arguably preferable to use the electronic channel for ballot return. Further, this enables voters to experience the full campaign before voting, and better addresses the human tendency toward procrastination. The primary concern with electronic return is security; something most commercial systems do not fully address. Remotegrity is an electronic-return voting system designed to provide secure and reliable transport, even in the presence of client-side malware, server compromise, or a corrupt EA.

End-to-End Verifiability. The use of cryptographic techniques to provide a verifiable tally while maintaining strong voter privacy has developed substantially since first proposed by Chaum in 1981 [5]. E2E polling place systems like Prêt à Voter [9] and Scantegrity [7] have been refined and are suitable for governmental elections [4,3]. E2E internet voting systems like Helios [1] and SCV [23] have been tested in binding student and organizational elections [2]. Helios is not designed to provide strong integrity when a voter's computer is malicious, and proof-of-concept vote-stealing malware has been proposed [14].

Client-side vulnerabilities can be addressed through a technique called *code-voting*, proposed by Chaum in 2001 [6]. With code-voting, voter choices are denoted with a set of random codes distributed to the voter out-of-band. Without knowledge of the codes, malicious devices cannot sensibly modify voter choices. Many proposals have refined this approach [16,18,17,24,19,26,15,25]. While these systems protect the voter from client-side vulnerabilities, they do not protect against a malicious EA (which knows all the codes), nor do they provide dispute resolution (see below). Remotegrity extends the code voting approach to satisfy these additional security properties.

[2] `http://www.fvap.gov/resources/media/evswfactsheet.pdf`

The literature also addresses the tangential problem of coercion-resistance in the unsupervised, remote voting setting. This line of research originated with Juels *et al.* [20]. Recent improvements include more efficient tallying [27] and the use of panic passwords [12]. These systems all assume the voter votes on a trusted machine. By contrast, code voting does not address coercion. Addressing both threats simultaneously is an open problem.

Dispute Resolution. One less obvious property an E2E voting system should provide is *dispute-freeness* [21] (or accountability [22]). If the verification of some aspect of the election fails, implying an error or fraud, the voter should be able to demonstrate that it failed and which entity is responsible. With online voting, the EA cannot assume accountability for the state of voters' computers. If vote verification fails, the EA must ensure that it is not incorrectly blamed for compromised voter machines. Likewise, voters want assurance that a malicious EA cannot modify ballots and blame the voters' computers if the modification is detected. It is also important that voters or political parties cannot easily fabricate false evidence that an election has been compromised, casting doubt on the final tally.

3 Remotegrity

Overview. Remotegrity is not a full voting system. Rather, it is a component that is combined with a traditional E2E paper ballot system like Scantegrity or Prêt à Voter to provide integrity to the process of ballot delivery. Even when ballots are submitted from an untrusted computer over an untrusted network to an untrusted EA, voters can have the same assurance that their vote will be counted correctly as they would if they cast their ballot in-person.

It utilizes two primary security mechanisms. The first is code voting which prevents malicious devices from sensibly modifying voter selections. However this is not sufficient as a fully corrupt EA could determine the set of codes and modify voter selections reliably. The second mechanism we use is that of providing each voter with a *lock-in* code placed under a scratch-off surface. The lock-in code is posted on the election website by the voter to indicate that his or her vote is correctly recorded. The scratch-off surface operates as a tamper-evident seal. If a malicious EA locks in a ballot entry that does not reflect the voter's selections, the scratch-off surface still covers the code providing physical evidence of EA malfeasance.

3.1 Cryptographic Preliminaries

Remotegrity utilizes a distributed key generation protocol DKG to generate threshold shares of a secret seed s amongst a set of trustees (*e.g.,* party officials or election observers); a pseudo-random generator, $\mathsf{PRG}(s)$, to expand the seed into psuedo-randomness; and a cryptographic commitment function, $\mathsf{Comm}(m, r)$, that is hiding and binding for message m and randomness r (for brevity, we denote a randomized commitment to m as $[\![m]\!]$).

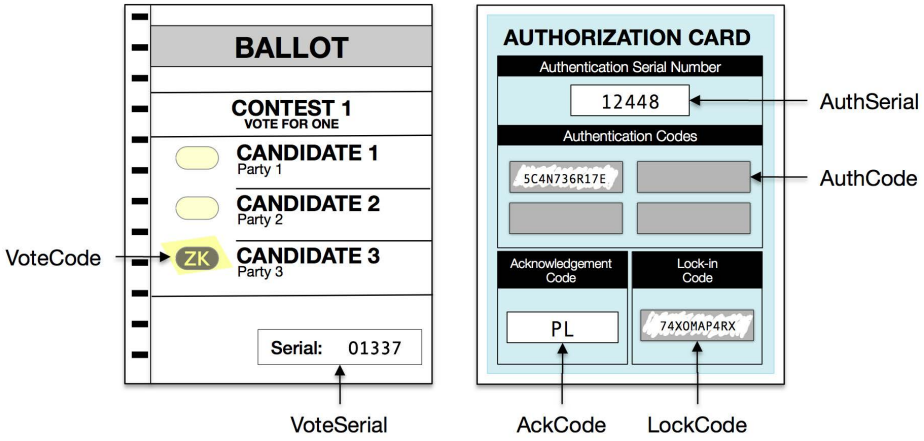

Fig. 1. Remotegrity ballot package. **Left:** marked Scantegrity II ballot showing a vote for candidate 3. **Right:** Remotegrity authorization card showing the AuthSerial and AckCode as well as an AuthCode and the LockCode as scratched off by the voter during the ballot casting protocol.

As in Scantegrity, we assume trustees can use a semi-trusted 'blackbox' computation to generate election values. This computation is not assumed to be correct, but it is assumed to keep all inputs and intermediate values private. No private state is ever stored; trustees always regenerate the state from their shares. The trade-off between the practicality offered by this model and the strong cryptographic guarantees of using a multiparty computation have been discussed elsewhere [13]. Finally we assume the existence of an append-only broadcast channel, called a bulletin board (BB).

3.2 Protocol

Voters receive a ballot package by mail which contains two parts, as shown in Figure 1. The first is a paper ballot, similar or identical to the ones used for polling place voting. In this section, we will consider composing Remotegrity with Scantegrity II ballots. Scantegrity II ballots consist of a serial number, VoteSerial, and a set of short confirmation codes, $\langle \mathsf{VoteCode}_1, \mathsf{VoteCode}_2, \ldots \rangle$. There is one code per candidate and the codes are randomly assigned to candidates and ballots. Two voters will, with high probability, receive different codes, invariant to whether they voted for the same candidate or different candidates. The codes are printed with invisible ink and revealed when the voter marks a particular candidate with a special pen (we describe how we modified the system to avoid having to mail pens to each voter in Section 4). For simplicity, we assume a single contest ballot in our description; extension to multi-contest ballots is trivial.

Ballot Casting

Each voter performs the following steps:

1. The voter enters the ballot and authorization card serial numbers $\langle \mathsf{VoteSerial}, \mathsf{AuthSerial} \rangle$ into the voting platform's user interface. The voting platform checks that neither serial number was previously posted to the BB.
2. Using the ballot, the voter selects the VoteCode appearing next their chosen candidate. Using the authorization card, the voter selects an AuthCode at random and to scratch-off. The voter enters the following information into the voting platform, which is posted by the platform to the BB:
   $\langle \mathsf{VoteCode}, \mathsf{AuthCode} \rangle$.

Upon receiving a new BB Entry, the trustees do the following:

3. The trustees check AuthCode. If it has not been used in a previously signed BB Entry and it contains valid codes, the trustees append AckCode and sign the tuple. The BB entry now reads:
   $\langle \mathsf{VoteSerial}, \mathsf{VoteCode}, \mathsf{AuthSerial}, \mathsf{AuthCode}, \mathsf{AckCode}, \mathsf{Sig}(\%) \rangle$,
   where $\mathsf{Sig}(\%)$ denotes a digital signature on all preceding elements in the tuple. If it does not contain valid codes, it marks it as invalid and signs it.

Upon receiving acknowledgement from the trustees, the voter does the following:

4. The voter checks that no modifications have been made to the BB Entry. The voter verifies AckCode and the signature. If correct, the voter submits LockCode. The BB Entry is now finalized as:
   $\langle \mathsf{VoteSerial}, \mathsf{VoteCode}, \mathsf{AuthSerial}, \mathsf{AuthCode}, \mathsf{AckCode}, \mathsf{Sig}(\%), \mathsf{LockCode} \rangle$.

After the election closes, the trustees do the following:

5. For the tuples containing a correct LockCode, the trustees input $\langle \mathsf{VoteSerial}, \mathsf{VoteCode} \rangle$ to the vote tallying system (*e.g.,* Scantegrity's BB).

Protocol 1. The vote casting procedure in Remotegrity

The second part of the ballot package is the Remotegrity authorization card. The card consists of a serial number, a set of authentication codes under scratch-off (denoted with a grey box), a short acknowledgement code, and a lock-in code under scratch-off. With *e.g.,* four authentication codes, the authorization card is denoted as:

$$\Big\langle \mathsf{AuthSerial}, \boxed{\mathsf{AuthCode}_1}, \boxed{\mathsf{AuthCode}_2}, \boxed{\mathsf{AuthCode}_3}, \boxed{\mathsf{AuthCode}_4}, \mathsf{AckCode}, \boxed{\mathsf{LockCode}} \Big\rangle$$

Election Set-up

Prior to the election, all trustees do the following with a blackbox computation:

1. The trustees use DKG to derive threshold shares of a master secret.
2. The trustees use PRG to expand the master secret into a sufficient number of random codes for two authorization cards per voter.
3. For each authorization card, the trustees publish on the BB the serial number and a commitment (again using PRG for the randomness) to each code on the card:

$$\langle \mathsf{AuthSerial}, [\![\mathsf{AuthCode}_1]\!], [\![\mathsf{AuthCode}_2]\!], \ldots, [\![\mathsf{AckCode}]\!], [\![\mathsf{LockCode}]\!] \rangle$$

After the pre-election commitments are published, the EA does:

4. The EA prints the authorization cards, potentially printing more than needed and allowing a random print audit of a fraction of the cards.
5. Each eligible absentee voter is assigned and mailed a Scantegrity ballot and an authorization card. The EA retains the binding between the voter ID, VoteSerial, and AuthSerial. For each ballot, it at least publishes: $\langle \mathsf{VoteSerial}, \mathsf{AuthSerial} \rangle$. The EA can also publish which voter received which VoteSerial without compromising ballot secrecy. In either case, the number of these tuples should match the number of absentee voters.

After the election closes, an authorized set of trustees open all the commitments to authorization card codes.

Protocol 2. The trustee and EA procedures in Remotegrity

Serials are assigned sequentially and all codes are assigned random; the length of the codes should provide resistance from repeated guessing (while "short" codes only resist a single guess). The purpose of each code is not likely apparent from inspection but each code and scratch-off surface plays an integral part in preventing certain attacks; thus we will explain the protocol concurrently to a security analysis. The vote casting process is described in Protocol 1, and how the codes are derived by the EA is described in Protocol 2.

Remotegrity protocol serves a single function: to allow voters to verify that their Scantegrity ballot, $\langle \mathsf{VoteSerial}, \mathsf{VoteCode} \rangle$, is correctly posted to Scantegrity's BB. If voters could post $\langle \mathsf{VoteSerial}, \mathsf{VoteCode} \rangle$ without interference from a client-side malware or a malicious EA, Remotegrity would not be required. The codes and features of the Remotegrity authorization card and vote casting protocol can be split into two sets. The first set contains the mechanisms for addressing a malicious voting platform: a single AuthCode and AckCode. The second set contains mechanisms for detecting malicious EA actions: multiple AuthCode's, LockCode, scratch-off surfaces, and the trustees signature.

Validating Ballot Codes. The protocol assumes that the EA can determine if a VoteCode for a given VoteSerial is valid: one of the VoteCode's appearing on the ballot. To provide certain assurances, Remotegrity uses the fact that a guessed VoteCode will, with high probability, be invalid. Scantegrity has its own dispute resolution process, which can determine precisely this. Assuming the systems are governed by the same set of trustees, they can work in an online fashion to validate the VoteCode in Remotegrity ballots as they are submitted. An alternative approach is append a short message authentication code to each VoteCode, which will be stripped off when the accepted and locked-in Remotegrity ballots are posted to Scantegrity's BB. This allows validation of the codes without requiring that all the confirmation codes be online and accessible to the trustees.

Initial BB Check. In the first step of Protocol 1, the voter checks if her VoteSerial has already been voted. If the VoteSerial appears but has been rejected by the EA for having an invalid AuthCode, the voter can ignore the entry and proceed to vote with an actual AuthCode. If the VoteSerial has been voted and accepted by the EA (*i.e.,* with a published AckCode and signature), it must have been posted by an insider with knowledge of the correct authorization code or the EA signed off on something invalid. In either case, the voter can demonstrate that no authorization codes have been scratched off on her card, which is publicly linked to the serial number of the ballot, and thus the EA is accountable for the wrongfully accepted ballot.

Malicious Voting Client. Provided the VoteSerial is not on the BB, we first consider the case where the EA is honest but the voter uses a malicious voting client. Since only the voter and the EA know the values of the codes on the ballot and authorization card, the voting client cannot cast a ballot without the voter's involvement or repeatedly guessing VoteCode and AuthCode pairs. Since VoteCode is short (*e.g.,* 2 characters), AuthCode should be of a length sufficient for protection from repeated guessing (*e.g.,* 12 characters).

When the voter enters VoteCode and AuthCode, the computer could keep AuthCode and modify VoteCode. It could further simulate the voter's view of the BB to make it appear that the BB Entry was not modified. To provide detection, the voter can rely on receiving back AckCode. Since the voting client does not know the VoteCode on the ballot corresponding to its preferred candidate, at best it can chose a VoteCode randomly. With moderately high (since the code is short) probability, the EA will reject the BB Entry and not post AckCode. The voting client will then have to guess AckCode which will also fail with moderate probability. Since receiving a wrong AckCode code suggests the computer is malicious, the client has only one chance to guess and thus AckCode can be short. Diligent voters can check the BB from a secondary device to detect modifications, even in the unlikely case that the computer issues a correct guess. If such detection occurs, the voter will not lock-in the ballot. Like AuthCode, LockCode should be of a length sufficient for protection from repeated guessing.

Malicious EA. We now consider a malicious EA. First, a point of clarification: a malicious EA could be comprised of colluding trustees who reconstruct the codes, the officials who print the authorization cards, or the officials who mail them. Since the EA is ultimately accountable for all of these officials, the Remotegrity protocol protects against all of them without distinguishing which exact official is responsible.

A malicious EA knows all of the codes on the voter's authorization card, however it cannot undetectably use a code unless it is assured the voter has scratched it off. Assume an EA generated/modified BB Entry is locked-in on the BB. If the voter did not try and lock something in, LockCode is still sealed and the voter can hold the EA accountable. If the voter has scratched-off LockCode, it must be the case that the voter's correct BB Entry did appear at some point on the BB and was accepted and signed by the EA. The EA cannot apply LockCode to any BB Entry other than the one intended by the voter without signing a new BB Entry. However, signing a new BB Entry requires the entry to have an unused AuthCode. Therefore, if the EA waits for the voter to submit LockCode and immediately fabricates a new BB Entry to which it applies LockCode, it would have to use a previously unused AuthCode and any unused AuthCode would still be sealed on the voter's authentication card.

Print Audit. Voters can resolve disputes by demonstrating that codes are still sealed on the physical ballots and authorization cards they have received. However, if the EA is forced to correctly commit to the contents of the cards, many disputes can be resolved without the physical records. In order to check this consistency, a random selection of authorization cards should be audited using a publicly verifiable challenge to determine the selection [11]. For full voter-verifiability, voters could be mailed two authorization cards: one to use and the one to audit.

3.3 Other Security Properties

Dispute Resolution. We say the EA *accepts* a BB Entry if it provides an AckCode and signs the BB Entry. If the EA accepts the BB Entry as cast by the voter, we call it a *true accept*. If it accepts a BB Entry that is modified from the voter's intent, or a BB Entry it manufactured without the voter's knowledge or consent, we call it a *false-accept*. If the EA rejects a BB Entry with correct values (*e.g.,* as a denial-of-service), we call it a *false reject*. Finally, if the EA correctly rejects a BB Entry containing incorrect codes (*e.g.,* one modified by a malicious computer, as above), we call it a *true reject*.

We iterate through all the various issues with each code and how it is resolved in Table 1. The EA can always force a denial-of-service, which is unsurprising as it can accomplish this without resorting to manipulating codes. What Remotegrity does not allow is the EA to fully accept (*i.e.,* accept and lock) any ballot the voter did not cast without the voter being able to dispute it.

If the voter enters values and does not see them on the BB, he or she tries again from another computer. All true rejects occur because the EA received

Table 1. Overview of the dispute resolution process in Remotegrity

Code	Issue	Blame	Resolution
VoteSerial	Missing	Device	Voter votes from a different device.
	False Accept	N/A	BB Entry belongs to another voter.
	False Reject	EA	Voter retains authentication card and ballot as evidence.
	True Reject	Device	Voter votes from a different device.
VoteCode	False Accept	EA	Voter attempts to change vote using another AuthCode.
	False Reject	EA	Voter retains ballot as evidence.
	True Reject	Device	Voter votes from a different device.
AuthSerial	False Accept	EA	Publicly apparent since link between VoteSerial and AuthSerial is public.
	False Reject	EA	Publicly apparent since link between VoteSerial and AuthSerial is public.
	True Reject	Device	Voter votes from a different device.
AuthCode	False Accept	EA	Voter retains unscratched AuthCode codes as evidence.
	False Reject	EA	Link between AuthCode and AuthSerial is decommitted after election.
	True Reject	Device	Voter votes from a different device.
AckCode	Invalid	Device	Voter accesses ABB from a different device.
Sig(%)	Invalid	EA	Publicly apparent. Voter can request new signature.
LockCode	False Accept	EA	Voter keeps unscratched LockCode as evidence.
	False Reject	EA	Voter retains authentication card as evidence.
	True Reject	Device	Voter locks-in from a different device.

false values. This happens because of a malicious voting computer or an erring human. If a voter sees false code(s) displayed on the BB and rejected by the EA, and knows it was not erroneously entered, he or she can attempt to enter the code(s) again from another computer. If, in spite of repeated attempts, the voter always experiences a similar reject, he or she is experiencing a distributed denial of service attack from voting computers.

A false reject occurs because an EA rejects a correct code claiming that it is incorrect; that is, the voter sees the correct code on the BB but the EA rejects it. The correspondence between AuthSerial and VoteSerial is public. Additionally, commitments to valid codes—all information on an authentication card; the correspondences between VoteCode and VoteSerial (though not between VoteCode and candidates)— are opened at the end of the election. Because the EA knows the correct correspondences, the EA is shown to be cheating. A voter may also experience a reject because of a previous use (not by the voter) of AuthSerial, VoteSerial, AuthCode, or LockCode or all—this would correspond to a previous false accept by the EA.

All false accepts are accepts of either (a) invalid codes or (b) valid codes (in either case, the accept is false because the code was not entered by the voter, but

can be seen on the BB). Case (a) is immediately apparent when the commitments for valid codes are opened, in a case converse to that described in false-rejects above. Because the EA knows an invalid code, its acceptance indicates a cheating EA and this is proven when the commitments are opened. For Case (b), if the false acceptance is of the VoteCode, the voter can try to re-enter the VoteCode from another computer. Because it is a short code, the computer might have guessed it correctly and used the correct VoteSerial, AuthSerial and AuthCode entered by the voter. For all other false-accepts—false accepts of LockCode or AuthCode—as well as repeated false accepts of the VoteCode, the voter should retain the unscratched-off authorization card and ballot to prove that the EA is cheating. (Here it is possible that a network of colluding dishonest voting computers would have guessed a VoteCode correctly and would repeatedly thwart the voter's attempt to change an incorrect VoteCode, but the probability is considered negligible). Note that incorrect correspondences between VoteSerial and AuthSerial are easily detected as being Case (a).

If the voter does not receive the correct AckCode, he or she attempts to vote again from another computer. Repeated failure implies an EA attempting a denial-of-service, assuming that the voter has access to at least one honest computer. This is proven when all the commitments are opened. If the voter receives an invalid signature, the entry is checked from a different computer. An invalid signature is apparent to anyone examining the BB.

Ballot Secrecy. No part of Remotegrity is dependent on the voter's selection. Secrecy of the voter's selection is fully subsumed by the Scantegrity system (or whatever E2E voting system Remotegrity is composed with). In particular, Scantegrity assumes that the printer can be trusted with knowledge of confirmation numbers, and that confirmation numbers printed in invisible ink are not visible unless exposed.

Physical Attacks on Scratch-Off Surfaces. Remotegrity does assume the integrity of scratch-off surfaces. If voters can retrieve codes without scratching-off the surface or can reapply an indistinguishable surface, they could falsely incriminate an entity for election tampering. The use of invisible ink and scratch-off is interchangeable. We present the ballots with invisible ink as per the original Scantegrity proposal, but use scratch-offs with Remotegrity as that is what was used in the election. Other physical technologies for providing tamper-resistant sealing of printed codes may be used with Remotegrity.

3.4 Optimizations

We avoid doubling-up the functionality of any of the codes to provide the clearest mapping between each code and the security functionality it serves. However to reduce the number of codes a voter must enter, codes can be combined. The serial numbers of the ballot and authorization card can be harmonized to the same value. If VoteCode and AuthCode are unique across all ballots/cards, serial numbers can be eliminated entirely. Finally, a unique AuthCode-length

code could be assigned to each candidate, eliminating the need for VoteCode at all. Note that this results in a fully-modified ballot style. Remotegrity is designed to interface with an existing type of ballot style, so that vote tallying can be conducted across all ballots: in-person and absentee together.

4 Deployment

Takoma Park is a municipality, sharing a city line with Washington D.C., with about 17,000 residents and 11,000 registered voters. The choice of voting system is formally made by the City Council, on recommendation by a Board of Elections (BOE) with 7 members. Ballots for municipal elections are provided in English and Spanish. Any voter can request to vote with an absentee ballot.

4.1 Preparations

We began discussion with the BOE in the early part of 2011 toward using Remotegrity in the November 2011 election. We attended their monthly board meetings and made many changes to the protocol based on their feedback.

System Test. The proposed system was tested on June 8, 2011 in the Takoma Park Community Center. The City announced the test in the city newspaper and in the senior newsletter. The test was open to anyone, and not restricted to Takoma Park voters or residents. We provided voters with a survey to fill out after they had tried the voting system. About 20 individuals participated in the test—including some BOE members—and about 17 responded to the survey on Remotegrity. From our perspective, the purpose of the test was to receive feedback on usability. It also served as an opportunity to educate potential voters on the system; as a result, we interacted significantly with voters using the system. We did not use the results as an indication of usability, due to the test's informality and small sample size, but the subjective feedback was very useful in making changes to the user interface and instructions. As one example, we modified the system so that voters did not need to enter both AuthSerial and VoteSerial; just AuthSerial.

The test was reported in the media and we presented the results to the BOE in the June meeting. The BOE outlined a number of concerns, centred around usability and security (because of the protocol's use of the internet, and the problems Washington DC had had with an internet voting trial [28]). In the July meeting, the Board members communicated to us that they had confidence in the technology, but they were concerned about the procedures, which appeared *ad hoc*, about potential security mishaps, and that the system had not been peer-reviewed. In this meeting, they communicated that they were leaning towards not using Remotegrity, but would go ahead with a mail-in Scantegrity ballot.

System Adaptations. In the August Board meeting, we proposed (to which they agreed) that the city provide voters with the option to use Remotegrity in addition to mailing back marked ballots. Only marked ballots would be counted, but voters using Remotegrity could test/"audit" the system, and, if they chose to lock-in their vote, could communicate that the system was accurately recording their vote. Instructions for "voting" and "auditing" would be sent in separate envelopes in the same package, with appropriate marking, so as not to overwhelm voters not interested in the audit.

Thus the system we finally used had some major differences with the protocol described in Section 3.2. Voters were not required to lock-in (this means that, in practice, an EA changing the vote using another AuthCode belonging to the voter could not be distinguished from the voter by a third party). Second, the Remotegrity system included ballots with visible codes (these ballots correspond to the original version of Scantegrity [8]). This avoids the requirement of mailing invisible ink development pens, however dispute resolution in the specific case of a wrong VoteCode is not possible. Third, voters needed to submit marked paper ballots by mail for votes to be counted; this eliminated any dependence on the internet, but made it possible for the EA to ignore a mailed-in ballot. However, the Scantegrity codes of the votes were posted on the election website and voters could check if their votes made it in the count.

4.2 Implementation and Server Infrastructure

Backend. The backend of the Remotegrity system contains a module, written in Java, that has similar functionality to the Scantegrity backend. Before the election, it is responsible for generating the Remotegrity data, commitments, and PDFs for printing the authorization cards. After the election, it is used to open the commitments.

Printing the Cards. The BOE anticipated that about 120 absentee voters would register. Because of the small-scale, around 200 authorization cards were printed by the Remotegrity team on a regular inkjet printer on card stock, and scratch-off stickers were applied manually. The back of each card had a printed overlay of "noise" to obfuscate the possibility of the reading of codes through the scratch-off surface using a very bright source of light.

Web-interface. The web interface was implemented with PHP and the Smarty template engine. During the election, the system was hosted in Amazon Elastic Cloud (EC2). It consisted of a load balancer that served the page over HTTPS[3], two instances of Apache servers (monitored in realtime, with auto-scale option), and one instance of an Amazon RDS (MySQL). Each server instance was only granted the right to INSERT data into the database. If needed, additional webservers could be started from the same image.

[3] `http://takoma.remotegrity.org`

Bulletin Board. Another EC2 instance ran a signing daemon written in Java. As data received from voters was inserted into the table by the webservers, the daemon would fetch and digitally sign it in realtime, inserting the signed data to a different table. This happened independently of the EA deciding to accept a ballot. Auditors had direct access to the second table.

An offline signing server (OSS) checked the validity of the submitted codes and was granted access to the AckCode codes corresponding to each possible AuthCode code. If the ballot submission was well-formed, it would sign it. As input, the OSS took an XML file containing data signed by the signing daemon, and output in XML an AckCode and signature on the entries it accepted. Both input and output files were transported to/from the OSS on a flash drive manually every 4 hours.

Testing. Both, the web interface and the backend of the system that was used during the pilot were tested by two independent researchers: Marco Ramilli and Marco Prandini. They found several security issues related to the web-interface, *e.g.,* visible system path and session control issues. These issues were fixed.

4.3 The Election

Voters were required to return their absentee ballots by mail, which still provides voters with the ability to verify correct receipt of their ballot (but limits their ability to respond and correct the ballot if it is not correct). In addition, they could opt-in to submitting their ballots electronically.

Procedure. Takoma Park election officials mailed two types of paper cards—a Scantegrity ballot and a Remotegrity authorization card—to each voter. Both cards were sent to the voter by regular mail in a single package. The ballots and authorization cards were paired at random and commitments to $\langle \mathsf{VoteCode}, \mathsf{AuthCode} \rangle$ were published. The EA assigned at random a package to a voter. They put the package into an outer envelope, stuck the voter's address on this envelope and wrote down the serial number of the authorization card next to a voter's name on a roster (this could help to remove a vote from the tally if it was intercepted by an unauthorized person and detected by a voter). Unused packages were later audited.

Result. The Remotegrity BB contains 123 entries which correspond to 119 voters. Only 5 ballots were submitted online, and two of these were not counted as the corresponding paper ballots were not mailed in. While the number of voters who used the online system was small, full preparation and a complete implementation were required to deploy the system.

Post-election. Remotegrity ballots were included in the same tally as Scantegrity ballots that were cast during election day. Both aspects of the election were audited by independent voting system experts selected by Takoma Park (on recommendation of the Remotegrity/Scantegrity teams). Neal McBurnett

and Roberto Araujo conducted the audit, which included verifying the Remotegrity commitments during the pre- and post-election audit procedures. Neal McBurnett additionally audited all the unsent absentee packages.

5 Lessons Learned and Concluding Remarks

The design of secure internet voting systems is non-trivial. One of the most important lessons learned concerns the importance of a good working relationship between the system designers and the election officials. We benefitted from Takoma Park's feedback on the user interface. We believe that they, in turn, came to appreciate some of the more subtle security properties we were attempting to provide, and that their involvement helped to promote an increased sense of pride and ownership of the outcome. The other important lesson pertains to adapting voting research systems for real-world use. For example, most E2E schemes presuppose the existence of a public append-only bulletin board. Implementing this, however, proved to be a major technical challenge, which invariably leads to a relaxation of security properties. Designers of such systems must be able to adapt accordingly.

In future work, while considering scalability of the system for larger elections, we do not foresee problems and observe that it is as scalable as vote-by-mail. Also interesting from the perspective of future work is the problem of rigorous definitions and property proofs for the protocol, in a model that takes into account the properties of paper and scratch-off surfaces. Another important open problem is that of a coercion-resistant version of Remotegrity.

Finally, it was exciting to work with an election jurisdiction that sees merit in cryptographic election verification. But this was not just a case of early adoption—Takoma Park had run an E2E election before, and for the first time, we caught an exciting glimpse into the future of electronic voting in which E2E verification is the new normal.

Acknowledgements. The authors acknowledge the contributions of the voters of Takoma Park, the City Clerk, the Assistant City Clerk, and all Board of Elections members. We would like to thank Ron Rivest for his valuable remarks. The authors thank Zbigniew Golebiewski for implementing OSS and moving data between online and offline instances, sometimes in the middle of the night.

References

1. Adida, B.: Helios: web-based open-audit voting. In: USENIX Security (2008)
2. Adida, B., de Marneffe, O., Pereira, O., Quisquater, J.J.: Electing a university president using open-audit voting: Analysis of real-world use of Helios. In: EVT/WOTE (2009)
3. Burton, C., Culnane, C., Heather, J., Peacock, T., Ryan, P.Y.A., Schneider, S., Srinivasan, S., Teague, V., Wen, R., Xia, Z.: Using Pret a Voter in Victoria State elections. In: EVT/WOTE (2012)

4. Carback, R.T., Chaum, D., Clark, J., Conway, J., Essex, A., Hernson, P.S., Mayberry, T., Popoveniuc, S., Rivest, R.L., Shen, E., Sherman, A.T., Vora, P.L.: Scantegrity II election at Takoma Park. In: USENIX Security (2010)
5. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 24(2), 84–90 (1981)
6. Chaum, D.: Surevote: Technical overview. In: WOTE (2001)
7. Chaum, D., Carback, R., Clark, J., Essex, A., Popoveniuc, S., Rivest, R.L., Ryan, P.Y.A., Shen, E., Sherman, A.T.: Scantegrity II: end-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In: EVT (2008)
8. Chaum, D., Essex, A., Carback, R., Clark, J., Popoveniuc, S., Sherman, A.T., Vora, P.: Scantegrity: End-to-end voter verifiable optical-scan voting. IEEE Security and Privacy 6(3), 40–46 (2008)
9. Chaum, D., Ryan, P.Y.A., Schneider, S.: A practical voter-verifiable election scheme. In: di Vimercati, S.d.C., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 118–139. Springer, Heidelberg (2005)
10. Clark, J., van Oorschot, P.: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In: IEEE Symp. Security & Privacy (2013)
11. Clark, J., Hengartner, U.: On the use of financial data as a random beacon. In: EVT/WOTE (2010)
12. Clark, J., Hengartner, U.: Selections: Internet voting with over-the-shoulder coercion-resistance. In: Danezis, G. (ed.) FC 2011. LNCS, vol. 7035, pp. 47–61. Springer, Heidelberg (2012)
13. Essex, A., Clark, J., Hengartner, U., Adams, C.: Eperio: Mitigating technical complexity in cryptographic election verification. In: EVT/WOTE (2010)
14. Estehghari, S., Desmedt, Y.: Exploiting the client vulnerabilities in internet e-voting systems: Hacking Helios 2.0 as an example. In: EVT/WOTE (2010)
15. Heiberg, S., Lipmaa, H., van Laenen, F.: On E-vote integrity in the case of malicious voter computers. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 373–388. Springer, Heidelberg (2010)
16. Helbach, J., Schwenk, J.: Secure internet voting with code sheets. In: Alkassar, A., Volkamer, M. (eds.) VOTE-ID 2007. LNCS, vol. 4896, pp. 166–177. Springer, Heidelberg (2007)
17. Helbach, J., Schwenk, J., Schage, S.: Code voting with linkable group signatures. In: EVOTE (2008)
18. Joaquim, R., Ribeiro, C.: CodeVoting protection against automatic vote manipulation in an uncontrolled environment. In: Alkassar, A., Volkamer, M. (eds.) VOTE-ID 2007. LNCS, vol. 4896, pp. 178–188. Springer, Heidelberg (2007)
19. Joaquim, R., Ribeiro, C., Ferreira, P.: VeryVote: A voter verifiable code voting system. In: Ryan, P.Y.A., Schoenmakers, B. (eds.) VOTE-ID 2009. LNCS, vol. 5767, pp. 106–121. Springer, Heidelberg (2009)
20. Juels, A., Catalano, D., Jacobsson, M.: Coercion-resistant electronic elections. In: WPES (2005)
21. Kiayias, A., Yung, M.: Self-tallying elections and perfect ballot secrecy. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 141–158. Springer, Heidelberg (2002)
22. Kusters, R., Truderung, T., Vogt, A.: Accountability: Definition and relationship to verifiability. In: CCS (2010)

23. Kutyłowski, M., Zagórski, F.: Scratch, Click & Vote: E2E voting over the internet. In: Chaum, D., Jakobsson, M., Rivest, R.L., Ryan, P.Y.A., Benaloh, J., Kutylowski, M., Adida, B. (eds.) Towards Trustworthy Elections. LNCS, vol. 6000, pp. 343–356. Springer, Heidelberg (2010)
24. Oppliger, R., Schwenk, J., Lohr, C.: Captcha-based code voting. In: EVOTE (2008)
25. Popoveniuc, S.: Speakup: remote unsupervised voting. In: ACNS (2010)
26. Ryan, P.Y.A., Teague, V.: Pretty good democracy. In: Christianson, B., Malcolm, J.A., Matyáš, V., Roe, M. (eds.) Security Protocols 2009. LNCS, vol. 7028, pp. 111–130. Springer, Heidelberg (2013)
27. Spycher, O., Koenig, R., Haenni, R., Schläpfer, M.: A new approach towards coercion-resistant remote e-voting in linear time. In: Danezis, G. (ed.) FC 2011. LNCS, vol. 7035, pp. 182–189. Springer, Heidelberg (2012)
28. Wolchok, S., Wustrow, E., Isabel, D., Halderman, J.A.: Attacking the washington, D.C. Internet voting system. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 114–128. Springer, Heidelberg (2012)