The New South Wales iVote System: 
Security Failures and Verification Flaws 
in a Live Online Election 

J. Alex Halderman and Vanessa Teague

1 University of Michigan  
jhalderm@eecs.umich.edu  
2 University of Melbourne  
vjteague@unimelb.edu.au

Abstract. In the world’s largest-ever deployment of online voting, the iVote Internet voting system was trusted for the return of 280,000 ballots in the 2015 state election in New South Wales, Australia. During the election, we performed an independent security analysis of parts of the live iVote system and uncovered severe vulnerabilities that could be leveraged to manipulate votes, violate ballot privacy, and subvert the verification mechanism. These vulnerabilities do not seem to have been detected by the election authorities before we disclosed them, despite a pre-election security review and despite the system having run in a live state election for five days. One vulnerability, the result of including analytics software from an insecure external server, exposed some votes to complete compromise of privacy and integrity. At least one parliamentary seat was decided by a margin much smaller than the number of votes taken while the system was vulnerable. We also found fundamental protocol flaws, including vote verification that was itself susceptible to manipulation. This incident underscores the difficulty of conducting secure elections online and carries lessons for voters, election officials, and the e-voting research community.

1 Introduction

Internet voting has rarely been used in significant elections for public office, due to numerous, well established security risks [12], such as compromise of election servers, of voters’ client devices, of the network in between, and of the voter authentication process. To better understand how these risks can play out in real elections, we studied what may be the largest deployment of Internet voting to-date, the March 2015 state election in New South Wales, Australia.

In this election, voters had the option to use an online voting system called iVote, which was developed by e-voting vendor Scytl in partnership with the New South Wales Electoral Commission (NSWEC). Prior to the election, NSWEC performed multiple security studies (e.g. [21][22]), and officials publicly claimed
that the vote was “...completely secret. It’s fully encrypted and safeguarded, it can’t be tampered with” \cite{Haldeman}. Over 280,000 votes were returned through iVote (about 5% of the election total), exceeding the 70,090 Norwegian votes submitted online in 2013 \cite{Norway} and the 176,491 online votes in the 2015 Estonian election \cite{Estonia}.

While the election was going on, we performed an independent, uninvited security analysis of public portions of the iVote system. We discovered a critical security flaw that would allow a network-based attacker to exploit the recently discovered FREAK attack \cite{FREAK} to defeat TLS and inject malicious code into browsers during voting, and we showed that an attacker could exploit this flaw to violate voter privacy and steal votes. We also identified several methods by which an attacker could defeat the verification mechanisms built into the iVote design.

After we reported these problems to authorities, NSWEC patched iVote to correct the network security flaw, but by this time the election had been running for five days and 66,000 votes had been cast on the vulnerable system. After the vulnerability was removed, we made our findings public in a technical blog post on Freedom to Tinker \cite{FreedomToTinker} and an essay for nontechnical readers in The Conversation \cite{Conversation}.

The election count is now complete \cite{Completion}, with the final seat in the proportionally-represented Legislative Council having come down to a margin much smaller than the number of votes cast over iVote before it was patched. To our knowledge, this is the first time enough votes to affect a parliamentary seat in a state election have been returned over an Internet voting system while it was demonstrably vulnerable to attacks that would allow external vote manipulation. While we do not know whether anyone exploited the opportunity for electoral fraud, we know the opportunity was there.

In this paper, we detail our security findings about iVote and draw broader lessons from this case study. The iVote vulnerability reinforces findings of security problems in other proposed and fielded Internet voting systems, such as Washington, D.C.’s \cite{Washington} and Estonia’s \cite{Estonia}, and it demonstrates once again that no amount of pre-election review can guarantee that such a system is secure. These problems also highlight the brittleness of the web platform and TLS protocol—a fragility which may be incompatible with the intensive security requirements and time pressure of political elections.

iVote’s failure should also encourage skepticism towards other Internet voting systems claimed to be verifiable. Years of research on electronic methods of election verification are only just beginning to produce end-to-end verifiable voting systems appropriate for use in low-stakes, low-coercion elections \cite{EndToEnd}, or in government elections using a paper mail step \cite{PaperMail}. The iVote verification protocol was not made publicly available for independent analysis, and clearly ignores basic insights and central techniques from that research. An election verification protocol, like any other security protocol, should not be relied upon without an extensive period of public review. The decision to use a telephone-based vote reading service instead of a genuine peer-reviewed cryptographic verification mechanism substantially
reduced voter privacy while ultimately providing only very limited assurances of integrity.

Securing Internet voting requires overcoming some of the most difficult problems in computer security, and, with existing technology, even the smallest mistakes can undermine the integrity of the election result. The experience in New South Wales is a concrete example demonstrating security problems that many security researchers, including us, have warned about for many years. We recommend that election officials refrain from conducting high-stakes elections online until there are fundamental security advances.

2 iVote Background

The iVote voting system is a complex interaction of many components, some managed by the NSWEC and some by other administrators. The system was implemented by Scytl, (a multinational electronic voting vendor), using a combination of custom and off-the-shelf code.

iVote registration and voting could each be done by any of three different methods: by telephone, by (remote) Internet, and by a NSWEC computer supplied in a polling place. There were four steps in a voter’s iVote experience:

1. The voter registered, received her 8-digit iVote ID number and chose a 6-digit PIN.
2. The voter logged in to the voting server (or the telephone voting server) with her iVote ID and PIN, cast a vote and received a 12-digit Receipt Number. The vote was encrypted on the client, returned to the voting server, and forwarded to the Verification Server.
3. (Optionally) the voter telephoned the Verification Server, an Interactive Voice Response system. She entered her iVote ID, PIN and Receipt Number, and heard her vote read back to her. This service was turned off at close of polls.
4. (Optionally) the voter logged into the Receipt Server, entered her receipt number, and received a statement of whether any votes with that receipt number were included in the final count. This service remained active after the close of polls.

More details are described in the Security Implementation Statement [22] and other reports published by NSWEC [18]. These include prose descriptions of the methods of encrypting and processing the vote. These documents describe procedures for a “Decryption Verification Ceremony”, in which an auditor could reconcile the votes on the verification server with the main election data. There are some procedures for removing duplicates and votes that were cast via a different channel, and for testing by running dummy votes through the process during the election and then removing them afterwards.

No source code is available for any of the server-side processes, including the auditor, main voting web server, verification server, registration server or receipt server. However, there is of course the opportunity to inspect the HTML and JavaScript delivered to the browser. The design has evolved over several different
drafts, which differ in some important respects from what the code actually does—see Section 5.

In the 2015 state election, each voter could cast one vote for the Legislative Assembly and one for the Legislative Council. Although iVote was officially reserved for the disabled and other eligible absentee voters, voters could qualify by self-certifying that they would be out of the state during election day \cite{19}. iVote opened to the public on the morning of March 16 and closed at 6 P.M. on March 28, the same time as all other polls closed in the general election. Officials reported that about 280,000 votes (5\% of all counted ballots) were cast using the iVote system.

This is the second state election in which NSWEC has run Internet voting, having used an earlier version (provided by Everyone Counts) in their 2011 state election. We have offered them over the years explanations of the general risks associated with Internet voting and specific analysis of problems in the protocols proposed for the 2015 state election. Very little of this advice was acted upon. When we made our recent findings public (after the vulnerability was removed) NSWEC responded by denying the seriousness of the problem, attacking us in the press and making a formal complaint to the University of Melbourne claiming a “breach of [the university’s] policy on freedom of speech.” We would have much preferred to persuade NSWEC not to run Internet voting, than to have discovered that 66,000 votes were cast while iVote was vulnerable to manipulation and privacy breach.

3 Finding vulnerabilities in iVote

Shortly after iVote voting opened, we began an independent security review of the publicly accessible components of the system. Although election officials did not publish the source code, client-side portions of this code were necessarily delivered to voters’ browsers. Since we were not eligible voters, we did not proceed past the login screen of the voting web application, https://cvs.ivote.nsw.gov.au, but we did inspect the HTML, CSS, and JavaScript code that made up the application. In addition, NSWEC made a practise version of the iVote system available to the public at large at https://practise.ivote.nsw.gov.au. The practise site allowed anyone to log in using provided credentials and vote a mock ballot. We confirmed that the practise system used substantially the same client-side code as the real election server and used it to perform further hands-on tests.

iVote is designed to deliver the web application from servers using HTTPS. This is intended to prevent an adversary subverting iVote by modifying or replacing the official code as it is transmitted via the Internet to the user’s web browser. However, not all HTTPS servers are secure—there are many configuration and operational details that system administrators need to get right in order to ensure that the protocol provides the desired security guarantees. We tested the security of the main iVote HTTPS server using a standard tool, the Qualys SSL Labs SSL
The New South Wales iVote System: Security Failures

Fig. 1. Like most web applications, iVote was made up of dozens of resources that were loaded in the background by the browser. Using the Chrome Developer Tools, we could see that most of the iVote resources came from the “core voting system” server, cvs.ivote.nsw.gov.au, but one component, JavaScript for the Piwik analytics tool, was loaded from an external server, ivote.piwikpro.com.

Test. The results indicated that the server configuration complied with current best practices and was secure against known vulnerabilities.

However, a closer analysis of the structure of the iVote application showed that one of the resources loaded by the site came from an external web server. When the voter loads the iVote site, the site imports and executes JavaScript for a third-party analysis tool called Piwik. As shown in Figure 1, this code is loaded from a URL at the third-party server https://ivote.piwikpro.com. We tested the SSL configuration of this site, we found that it was extremely poor—scoring an ‘F’ grade in the SSL Labs test, as shown in Figure 2. Among other security problems, the server used insecure Diffie-Hellman parameters, allowed 512-bit export cipher suites that are subject to the FREAK attack, and was vulnerable to the POODLE attack. This weak configuration allows multiple ways for an attacker to bypass the security provided by SSL and inject malicious code into the user’s iVote session without triggering any browser security warnings.
Fig. 2. The ivote.piwikpro.com server scored an F on the Qualys SSL Labs tests. Among other problems, the server used insecure Diffie-Hellman parameters, allowed 512-bit export cipher suites that are subject to the FREAK attack, and was vulnerable to the POODLE attack. We showed that these problems would allow a man-in-the-middle attacker to inject vote-stealing code into the iVote application.

3.1 Background on the FREAK attack

We found that it would be possible for a network-based man-in-the-middle attacker to compromise iVote by exploiting the FREAK attack against the vulnerable Piwik server. The FREAK attack \[5,9\], short for Factoring RSA Export Keys, is a TLS vulnerability that was publicly disclosed on March 3, 2015, less than two weeks before the start of the election. As the name implies, FREAK exploits the weakness of 512-bit “export-grade” RSA keys that are supported by the TLS protocol as a legacy feature of 1990s era U.S. cryptographic export restrictions. If a server supports export-grade RSA—as did ivote.piwikpro.com—an attacker can fool many browsers into using this reduced-strength cryptography, obtain the RSA private key by factoring the 512-bit public key, and manipulate the contents of the connection.

The attack begins by intercepting the client hello message and sending a substitute client hello to the server declaring that the client wishes to use export-grade RSA. In 512-bit RSA key-exchange modes, the TLS server sends a 512-bit “temporary” RSA public key to the client and signs this key, together with a
nonce chosen by the client, using the public key from its normal X.509 certificate. The client verifies that there is a valid chain of certificates from the server’s X.509 certificate to a trusted root certificate authority, then uses the temporary RSA key to encrypt session key material that will be used to secure the remainder of the connection. The FREAK attack exploits a mistake in the way browsers process the server’s message containing this temporary key. Several widely used TLS implementations would accept a temporary export-grade RSA key even if the client did not ask for it. This allows the attacker to downgrade a connection requesting normal RSA encryption to much weaker export-grade RSA.

To exploit the FREAK attack against iVote, the attacker would have to be a network-based man-in-the-middle, with the ability to intercept and manipulate traffic destined for the Piwik server. This could be done, for instance, by compromising routers or WiFi access points, by poisoning DNS caches, or by having entirely legitimate administrator privileges on a home, workplace or ISP router. (Man-in-the-middle attacks are, of course, one of the main threats that HTTPS is designed to guard against.)

### 3.2 Exploiting the FREAK attack

At this point, the attacker needs to convince the user’s browser that he is really ivote.piwikpro.com. To do this, he needs the server’s signature on the client’s TLS nonce and an RSA public key that he knows the private key for. Assume for now that Piwik always uses the same 512-bit key. Nadia Heninger has shown that it is possible to factor 512-bit RSA keys using open-source software and Amazon EC2 in about 7 hours at a cost of about $100. Once the attacker has factored the key, he can intercept the user’s connection, note the client’s nonce, and make a request to the real Piwik server with the same nonce—in effect, using it as a signature oracle. He can send the resulting signature on the RSA key as part of the connection to the voter’s browser, which will see the key as valid and use it to encrypt its session key material. Since the attacker has factored the key, he can decrypt this key material and impersonate the Piwik server for the rest of the connection.

One complication is that the Piwik server, unlike many TLS implementations, periodically rotated its temporary key. In our tests, we saw the key change approximately every hour—too frequently to apply simple factoring methods available to us. However, we found that we could force the Piwik server to use the same temporary RSA key for much longer periods by maintaining a long-lived TLS connection and repeatedly invoking client-initiated renegotiation. Each renegotiation can use a different client nonce, so, by using this method, we could use the Piwik server as a signature oracle to attack as many clients as we wanted and use the same key for as long as this connection stayed open.

In tests, we were able to sustain the connection for 17–21 hours, and, with Nadia’s assistance, we factored the temporary RSA key from one such session. An attacker could start such a connection, spend the first 7 hours factoring the key, and then attack an unlimited number of voters’ TLS connections for the remainder of the connection lifetime. By making multiple such connections in a
Fig. 3. Although the NSW web server used a secure HTTPS configuration to deliver the iVote application, the app subsequently loaded additional JavaScript from an insecure external server, ivote.piwikpro.com. An attacker who intercepted connections between the voter’s browser and the PiwikPro server could tamper with this JavaScript to inject arbitrary malicious code into the iVote application.

In a staggered fashion, the attacker could have continuously attacked iVote users for the duration of the connection at a cost of about $100 per 12-hour period.

3.3 Proof-of-concept attack

We developed a proof-of-concept demonstration to show how an attacker could leverage the FREAK vulnerability to manipulate the iVote system. Following the overall scheme in Figure 3, this attack exploited the vulnerabilities in the Piwik server to replace the code loaded from ivote.piwikpro.com with malicious JavaScript. Since this code was executed in the context of the user’s iVote session, it could arbitrarily change the operation of the iVote web application.

The demonstration malicious code we injected hooked into key parts of the iVote client code. iVote used AngularJS to run a series of worker JavaScript threads which implemented cryptographic operations. Crucial election data, including the contents of the vote, were passed between these workers as messages. Our code intercepted these messages to change the intended vote to a different vote as it is passed to the worker script that performed the encryption. This changed the vote that was sent to the iVote server. Our code also exposed the vote that the voter intended to cast and sent it, along with the voter’s authentication credentials, to a command-and-control server operated by the attacker. Screenshots from our demonstration are in Figure 4.
Fig. 4. As a proof of concept, we showed that we could exploit the FREAK attack against iVote to inject malicious code that would surreptitiously manipulate the voter’s choices (left) and report them to a command-and-control server (right). Our mock attacker’s symbol invokes Ned Kelly, an iconic Australian outlaw.

Since we (of course) would not attempt to steal actual votes, we tested this demonstration attack against the iVote practice system, which was identical in all relevant respects to the real voting system. We notified the Australian CERT of this vulnerability around 2 P.M. on Friday, March 20. CERT took responsibility for notifying the NSW Electoral Commission, which fixed the problem around midday on Saturday, March 21, by modifying the iVote server configuration to disable Piwik. By then, about 66,000 votes had already been cast.

Most popular browsers were vulnerable to FREAK, including Internet Explorer, Safari, and Chrome for Mac OS and Android. Although patches were released for most browsers around March 10, iVote voting opened on March 16, and many users likely had not applied the relevant patches. We cannot know with certainty what fraction of iVote voters were vulnerable to this attack, or what other possible as-yet-unidentified problems might have had a similar effect.

4 Verifiability

The vote manipulation attack would be detected with some probability by the verification mechanism. However, the verification mechanism itself suffers from a number of straightforward circumventions and at least one important protocol flaw.

4.1 Simple verification avoidance

Telephone-based verification is easily sidestepped for last-minute votes because it shuts down at close of polls. (This closure seems to have been implemented in recognition of the threats to vote privacy associated with the Verification Server’s simultaneous knowledge of the voter’s iVote ID, vote contents and possibly telephone number.) So a compromised client could confidently modify votes
that were cast immediately before the deadline, knowing that they could not be verified. A malicious client (or server) could slow down near the end of polling to exacerbate this problem.

Voters are told how to verify by the same website they use to vote, so the attacker could direct the voter to a fake verification phone number that would read back the voter’s intended choices. Thanks to modern VoIP technology, setting up an automated phone system is just a matter of software.

The attacker could delay submitting the vote and displaying the receipt number for a few seconds, in hopes that the voter didn’t intend to verify and simply left the website. (Perhaps the site could show a progress bar in place of the number.) If the voter navigated away, there would be no chance to verify, and the attacker could confidently submit a fraudulent vote. Otherwise, the attacker would give up, submit the voter’s genuine vote, and display the receipt number.

4.2 Using the “clash” attack to reduce verification failures

The following attack allows an attacker who has intercepted many iVote sessions to share information between them and hence manipulate a large number of votes with very limited detection. The attack is a variant of the “clash” attack [14]. We believe it would work, but of course we were unable to test it without interfering with real votes.

When verification fails to produce the expected vote, the voter is supposed to complain to the authorities. Inevitably, some voters will falsely complain, either mistakenly or maliciously, that their correctly-entered vote has been dropped or misrecorded. The iVote verification design doesn’t provide any evidence to support or disprove voter complaints, making it difficult to distinguish an attack from the baseline level of complaints due to voter error. This observation is important in the following attack, which reduces the number of complaints, but probably doesn’t eliminate them altogether. Although this attack would sometimes be detected, the percentage of verification complaints could substantially underrepresent the fraction of manipulated votes.

First observe that, while the registration server itself was protected by SSL/TLS, the main iVote gateway from which voters reached it ran plain http. This gave a man-in-the-middle attacker the opportunity to misdirect registration attempts to a site of the attacker’s choosing, for instance by using the SSL strip attack [16].

This attack requires the ability to

– misdirect some registrations,
– compromise some clients, using for example the attack from Section 3 or simple misdirection, and
– give someone a PIN at registration, as opposed to letting them choose.

3 Or rather, it did for the first week of voting, before our recommendation to change it to SSL/TLS was adopted by NSWEC.
4 The real Registration Server asked the voter to generate a PIN (Security Implementation Statement p. 40), “a PIN is received from the elector by the Registration System.
First a brief note about Australian voting: Australian elections use multi-candidate preferential voting, so two voters who support party A may subsequently list quite different lower preferences. However, some common patterns recur very often, for example the vote consisting of a single (first) preference on each ballot. Many voters also follow official party “How to Vote” cards. Although we are not aware of data for NSW, studies in the neighboring state of Victoria show that overall about 40% of voters follow their how to vote card exactly [29]. We hope to be able to quantify this if NSW data become available.

The main idea of this attack is to intercept a voter’s registration and give him the iVote ID and PIN of a like-minded person who has already voted, preferably one who has cast a simple vote likely to be repeated. If the target voter’s choices exactly match those of the first voter, then all of the verification will look exactly right to both voters. The attacker can safely reuse the target voter’s registration credentials to get a new iVote ID and PIN and cast an arbitrary vote. If the target voter’s choices are different from the first voter, he will detect a problem if he calls the Verification Server, but not if he contacts the Receipt Server only.

This attack removes a party-A vote and substitutes a vote of the attacker’s choice. While it may sometimes be detected, if prediction of voter behavior is good then it raises far fewer complaints than that quantity of attacked votes ought to. For example, if prediction is perfect then it raises no alarm; if prediction is near-perfect then it manipulates many more votes than the number of verification complaints indicates. Note that it is not hard to predict how someone will vote when you have their registration credentials and hence their electoral roll record.

We find it interesting that issues mentioned in the academic literature on verifiable voting, including the absence of dispute resolution (or accountability) and the prospect of a particular kind of attack, turn out to be relevant in practice.

5 Other issues concerning Audit and Verification

It is impossible to tell whether there are other substantial risks to integrity, because the verification and audit systems are incompletely described and no source code is publicly available.

First observe that even the requirements for demonstrating integrity are much below those of an ordinary scrutineering process. A related compromise of the Core Voting System and the Verification Service could undetectably alter votes, because the Verification Service could simply lie to the voter about what vote was recorded on their behalf. Then the Core Voting System and Verification System could show consistent misrepresented votes to the Auditor.

A compromised web server or Voice Server (i.e. Server for voting by phone) could perform the attack from Section 4.1 on last-minute votes just as easily as a compromised client, knowing that the verification server will be turned off before there is an opportunity for the voter to detect this. (Note that this would require forging the “signing” of the vote, which is easy enough since it is based

---

The NSWEC then provides the elector with a unique electronic vote identifier (the iVote Number).
on javascript served by the server itself, and in the case of the telephone-voting server is performed by that server.)

The process for Auditing is only very vaguely described, so it is not clear whether a related compromise of the Core Voting System and the Auditor could undetectably alter votes. The attack would be simply for the auditor to turn a blind eye to inconsistencies between the Core Voting System’s data and the Verification Server’s. The Security Implementation Statement \cite{22} refers to some independent parties being allowed to observe some parts of the audit process, and receive some software, but it doesn’t say exactly what data they may check.

Votes that were present on the verification server (and possibly verified) could subsequently be removed if the voter re-registered or voted via another channel. We do not know how the auditor (let alone anyone else) verified that only the correct votes were removed.

Registering required only the voter’s name, date of birth and registered address. A postcard-based notification system was designed to notify voters in the case that someone else registered fraudulently on their behalf.

There are inconsistent descriptions of how votes are encrypted. The System Overview describes them as being encrypted with the Receipt Number; the Security Implementation Statement describes them as being encrypted using the ElGamal public key encryption system with the public keys of the Election and Verification Servers. Our inspection of the javascript used by iVote clients indicates that neither description is completely accurate: votes are encrypted using a “digital envelope” which consists of a randomly-generated symmetric key, encrypted once each with the Election and Verification Servers’ ElGamal public keys, plus the vote choices encrypted with AES using the symmetric key.

This is a sensible way of encrypting data when taken in isolation (indeed, much more sensible than the 10- or 12-digit symmetric keys that were first described), but its deviation from the specification has implications for both privacy and verifiability. The deviation of actual code from the publicly available specification begs the question of how the public can have confidence in the operation of processes whose source code is unavailable, particularly the Verification and Audit processes.

6 Privacy

First note that telephoning a third-party server to have a vote read back is unprecedented either in Australia or (to the best of our knowledge) elsewhere in the world. This introduces opportunities for privacy breaches via the third party and coercion of the individual after voting that do not exist in traditional paper-based voting. Such attacks could originate anywhere in the world and could even be automated—a coercer could simply offer money in return for iVote verification credentials that produced the desired vote from the verification server. Independent analysis has noted that the coercer could use the Receipt Server to check that the voter had not revoted \cite{17}. It is one thing to acknowledge that
remote voting is susceptible to coercion; it is quite another to add extra features that facilitate more coercion.

Furthermore, compared to paper (postal) ballots, for which the physical separation of the voter’s identity from the ballot occurs irrevocably, the destruction of electronic links is much more difficult to achieve. Various items of unique or private data are stored in various different parts of the iVote system—the Security Implementation Statement” makes reference to these being destroyed, without a clear understanding or description of how difficult it is to destroy electronic data irrevocably, especially if it has already been openly or illicitly observed, recorded or transferred elsewhere.

Encryption alone does not guarantee vote privacy. It is also important to prevent anyone from linking the decrypted vote back to the voter by tracing the encrypted vote through the system. Some electronic voting systems, including the Norwegian Internet voting system [11], use verifiable mixing in order to hide the link between the decrypted vote and the encrypted form submitted by the voter. The “cryptographic envelope” form of encryption used in iVote does not seem conducive to these methods. It is therefore crucial for privacy that the voter’s identity cannot be reconnected with her symmetrically-encrypted vote, which seems to remain in the same recognisable form throughout the process. Unfortunately, there are many ways this connection might possibly be made.

1. In the polling-site version of iVote, voters register and then vote via the same machine. This means that their identity and their vote are both present on the same machine.

2. Compromise of the Voice Server (i.e. the method of phone voting) could expose the contents of all votes cast via that method. This is particularly problematic given that many such voters will use identifiable phones.

3. The verification server has access to both identification and voting data.

4. All the phone communications, including voting and verifying by IVR, are potentially susceptible to eavesdropping, since the NSWEC has no control over the encryption (if any) used on the telephone line. Again, this is particularly serious since the plaintext vote is transmitted over a channel that could quite possibly identify the voter.

5. A compromise of the registration server, which knows the link between an individual’s iVote ID and name, could be combined with only one other (the Verification Server, Voice Server, or possibly Auditor) to link the name to the decrypted vote.

7 Other Problems

iVote suffered other problems during the election period, leading some parties to express informally their intention to challenge the result if it was unfavourable. Early in the voting period, the system was suspended for six hours because

---

5 Some also use homomorphic tallying, but that would not work for Australian (preferential) voting.
Fig. 5. iVote suffered from problems beyond security. Two parties were mistakenly left off the “above the line” section of the ballot for the first 19,000 votes. Some voters reported difficulty navigating the ballot, which required scrolling horizontally and vertically to access all 24 party groups and 394 candidates. Scroll bars failed to appear on some browsers, and the red arrows at the top had no effect. The “Continue” button with the right-pointing arrow ended the voting session and took the voter to a review screen.

two minor parties had been left off the “above the line” section of the ballot. The problem, blamed on human error, was fixed—but not before 19,000 votes had been cast. Other commentators drew attention to serious user interface problems [13] (See Figure 5).

8 Lessons

8.1 Security: the difficulty of correcting known problems in time, and unknown problems at all

The vulnerability to the FREAK attack illustrates once again why Internet Voting is hard to do securely. The system had been in development for years, but FREAK was announced only a couple of weeks before the election. New vulnerabilities are discovered regularly in software and protocols that an Internet voting system depends on for its security, including web browsers and TLS. When
this happens near election day, there may not be time to ensure that election servers and voters’ clients are properly retested and patched. Indeed, mechanisms for trying to ensure that the correct software is running conflict with the necessity for rapid patching. A last minute change could fix a serious security problem, or it could introduce an error or a deliberate attempt at fraud.

The specific vulnerability to the FREAK attack described above was eventually removed. However, almost every week brings a newly discovered security vulnerability, and it is impossible for NSWEC and Scytl to defend against attacks that may not even be publicly known until long after the election. We can bet that there are one or more major HTTPS vulnerabilities waiting to be discovered (and perhaps already known to sophisticated attackers).

8.2 Transparency: the importance of openly available source code, procedures and system details

The decision to keep the system’s source code and other design details secret until election time prevented security researchers from being able to alert the NSWEC to these specific problems before the election period.

We found a serious vulnerability and a number of protocol flaws by investigating only the small part of the system that was legally available for independent analysis. What other vulnerabilities might remain unnoticed in the large part of the system that is not available to the public?

8.3 Verifiability: when does an advertised verification mechanism truly provide verifiable evidence of a correct election outcome?

Although some schemes do provide genuine electronic election verification remotely, including Helios [2], Remotegrity [31], and Pretty Good Democracy [23], achieving this in a privacy-preserving way requires real verification work from the voter. Such techniques hold out promise for the future, and have been used successfully in elections with relatively educated voters and low stakes [3]. However, extending these techniques to state-level elections remains impractical for now. Issues such as voter authentication and usability remain problematic. (New South Wales has no public key infrastructure and requires voters to number multiple preferences on a ballot with 394 candidates.)

The iVote verification and audit protocols were not based on any peer-reviewed end-to-end verifiable scheme. It is hardly surprising that the verification mechanism was itself vulnerable to circumvention, since it had had no public peer review, did not protect vote privacy, and had only limited details and no source code publicly available.

There are sensible solutions for polling-place electronic election verification, including voter-verifiable paper trails with paper counting or risk limiting audits [15], and genuine end-to-end verifiable pollsite voting schemes [4, 7, 24].

When an Internet voting system is claimed to be verifiable, this claim should be supported with a clear argument based on openly available source code and
system and protocol details. Otherwise the verification protocol itself could be incomplete, erroneous, or open to manipulation.

8.4 Insider threats

It is worth noting that the most insidious vulnerabilities in this sort of system come from insiders, including the sort of insiders who may not obviously be in a position to violate ballot privacy or manipulate the results of the election. Many pieces of software contribute to an online (or poll site) voting experience. The decision to import javascript code from a third party into iVote opened up the possibility for that party to manipulate or expose votes. Analytics software has been known to expose critical information in online banking. Even if the piwikpro server had not been vulnerable to man-in-the-middle attacks by external attackers, anyone with (entirely legitimate) administrator privileges on their server would have been able to mount the same attack.

9 Conclusion

We found that an attacker could subvert the iVote voting session, expose the vote that voter intended to cast, substitute a different vote, and sidestep the verification mechanism so that last-minute manipulation was undetectable and “clash” attacks were under-detected. Implementing the attack required some skill but no special knowledge that was not publicly available at the time. Most voters who had not recently updated their web browser were vulnerable.

The incumbent conservative government has been comfortably re-elected. However, the final seat in the proportionally-represented Legislative Council was decided by a margin of at most 3177 votes, much smaller than the 66,000 votes cast over iVote during the time it was vulnerable to manipulation.

We reiterate our longstanding recommendation to the NSWEC and others to discontinue Internet voting. If Internet voting must proceed, future versions should have a genuine peer-reviewed verification mechanism, publicly available source code and documentation, a firm eligibility restriction to voters unable to vote via a more secure channel, and a clear public statement of the risks to vote privacy and electoral integrity.

Acknowledgments

The authors thank David Adrian, Ed Felten, Rajeev Goré, Nadia Heninger, Harri Hursti, and Liz Minchin for assistance during this project. For their support and encouragement after we made our results public, we would also like to thank a tremendous community of election integrity scholars and advocates, including but not limited to: Duncan Buell, David Dill, Joseph Hall, Candice Hoke, David Jefferson, Noel Runyan, Ronald Rivest, Barbara Simons and Pamela Smith.

6 See for instance http://oono.windytan.com/pankki.html
This material is based in part upon work supported by the U.S. National Science Foundation under grants CNS-1345254 and CNS-1409505, and by the Morris Wellman Faculty Development Assistant Professorship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. ABC News. Computer voting may feature in March NSW election, Feb. 2015. http://www.abc.net.au/news/2015-02-04/computer-voting-may-feature-in-march-nsw-election/6068290.
2. B. Adida. Helios: Web-based open-audit voting. In 17th USENIX Security Symposium, Aug. 2008. https://vote.heliosvoting.org.
3. B. Adida, O. De Marneffe, O. Pereira, and J.-J. Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of Helios. In USENIX/ACCURATE/IAVoSS Electronic Voting Technology Workshop (EVT), 2009.
4. S. Bell, J. Benaloh, M. D. Byrne, D. DeBeauvoir, B. Eakin, G. Fisher, P. Kortum, N. McBurnett, J. Montoya, M. Parker, et al. Star-vote: A secure, transparent, auditable, and reliable voting system. The USENIX Journal of Election Technology Systems, 1 (1), pages 18–37, 2013.
5. B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In 36th IEEE Symposium on Security and Privacy, 2015. To appear, https://www.smacktls.com/smack.pdf.
6. R. Carback, D. Chaum, J. Clark, J. Conway, A. Essex, P. S. Herrnson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, et al. Scantegrity II municipal election at takoma park: the first e2e binding governmental election with ballot privacy. In Proceedings of the 19th USENIX conference on Security, pages 19–19. USENIX Association, 2010.
7. D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. Sherman, and P. Vora. Scantegrity: End-to-end voter-verifiable optical-scan voting. Security & Privacy, IEEE, 6(3):40–46, 2008.
8. C. Culnane, P. Y. A. Ryan, S. Schneider, and V. Teague. vvote: a verifiable voting system. ACM Transactions on Information and System Security. To appear. Technical Report at http://arxiv.org/abs/1404.6822.
9. Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. Tracking the FREAK attack. https://freakattack.com/.
10. Estonian Internet Voting Committee. Statistics about Internet voting in Estonia, May 2014. http://www.vvk.ee/voting-methods-in-estonia/engindex/statistics.
11. K. Gjøsteen. The norwegian internet voting protocol. In E-Voting and Identity, pages 1–18. Springer, 2012.
12. N. Hastings, R. Peralta, S. Popoveniuc, and A. Regenscheid. Security considerations for remote electronic UOCAVA voting. National Institute of Standards and Technology, NISTIR 7770, Feb. 2011. http://www.nist.gov/itl/vote/upload/NISTIR-7700-feb2011.pdf.
13. A. Heber. There’s a huge design flaw in the NSW online voting system which Labor wouldn’t be happy about. Business Insider Australia, Mar. 28 2015. http://www.businessinsider.com.au/theres-a-huge-design-flaw-in-the-nsw-online-voting-system-which-labor-wouldnt-be-happy-about-2015-3

14. R. Kusters, T. Truderung, and A. Vogt. Clash attacks on the verifiability of e-voting systems. In 33rd IEEE Symposium on Security and Privacy, pages 395–409, 2012.

15. M. Lindeman and P. B. Stark. A gentle introduction to risk-limiting audits. IEEE Security and Privacy, 10(5):42, 2012.

16. M. Marlinspike. New tricks for defeating SSL in practice. Black Hat, 2009.

http://www.thoughtcrime.org/software/sslstrip/

17. R. McKay. Flaws in ivote’s re-vote process which attempts to defeat coercers. http://www.bigpulse.com/governmentalelections#changevoteflaw BigPulse.

18. NSW Electoral Commission. Index of iVote reports. http://www.elections.nsw.gov.au/about_us/plans_and_reports/ivote_reports

19. NSW Electoral Commission. iVote: Frequently asked questions. https://www.ivote.nsw.gov.au/faq.aspx

20. NSW Electoral Commission. NSW 2015 legislative council election - final distribution of preferences.

21. NSW Electoral Commission. iVote threat analysis and risk assessment, Jan. 2014. http://www.elections.nsw.gov.au/__data/assets/pdf_file/0008/175760/NSW_Election-_iVote_Threat_Analysis_and_Risk_Assessment_v3.0.pdf

22. NSW Electoral Commission. iVote system security implementation statement, Mar. 2015. http://www.elections.nsw.gov.au/__data/assets/pdf_file/0007/193219/iVote-Security_Implementation_Statement-Mar2015.pdf

23. P. Y. Ryan and V. Teague. Pretty good democracy. In Security Protocols XVII, pages 111–130. Springer, 2013.

24. P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia. Prêt ` a Voter: A voter-verifiable voting system. IEEE Transactions on Information Forensics and Security, 4(4):662–673, 2009.

25. B. Segaard, D. A. Christensen, B. Folkestad, and J. Saglie. Internettvalg: Hva gjør og mener velgerne?, 2014. https://www.regjeringen.no/globalassets/upload/kmd/komm/rapporter/isf_internettvalg.pdf

26. D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman. Security analysis of the Estonian Internet voting system. In 21st ACM Conference on Computer and Communications Security (CCS), Nov. 2014.

27. V. Teague and J. A. Halderman. Security flaw in New South Wales puts thousands of online votes at risk. Freedom to Tinker blog post, Mar. 22 2015. https://freedom-to-tinker.com/blog/teaguehalderman/ivote-vulnerability/

28. V. Teague and J. A. Halderman. Thousands of NSW election online votes open to tampering. The Conversation, Mar. 23 2015. https://theconversation.com/thousands-of-nsw-election-online-votes-open-to-tampering-39164

29. Victorian Electoral Commission. Report to Parliament on the 2010 Victorian State election; Section 11: Statistical overview of the election, 2011. http://www.vec.vic.gov.au/files/ER-2010-Section11.pdf

30. S. Wolchok, E. Wustrow, D. Isabel, and J. A. Halderman. Attacking the Washington, D.C. Internet voting system. In 16th International Conference on Financial Cryptography and Data Security (FC), Feb. 2012.

31. F. Zagoński, R. T. Carback, D. Chaum, J. Clark, A. Essex, and P. L. Vora. Retmotegrity: Design and use of an end-to-end verifiable remote voting system. In Applied Cryptography and Network Security (ACNS), pages 441–457. Springer, 2013.