



Review Your Choices: When Confirmation Pages Break Ballot Secrecy in Online Elections

James Brunet¹, Athanasios Demetri Pananos², and Aleksander Essex³

Western University, London, Canada
{jbrunet8, apananos, aessex}@uwo.ca

Abstract. Online voting systems typically display a confirmation screen allowing voters to confirm their selections before casting. This paper considers whether a network-based observer can extract information about voter selections from the length of the exchanged network data.

We conducted a detailed analysis of the Simply Voting implementation, which had randomly varying lengths of exchanged data due to dynamic page content and gzip compression. We demonstrated that we could correctly guess a voter's selection with accuracy values ranging up to 100% in some instances. Even on more complex ballots, we generally could still rule out some combinations of candidates. We conducted a coordinated disclosure with the vendor and worked with them to roll out a mitigation.

To their credit, this discovery (and therefore its fix) was made possible by their willingness to provide a *publicly* accessible demo, which, as we will show, remains a rarity in the industry.

Keywords: Ballot secrecy · TLS · Privacy · Online voting

1 Introduction

Online voting is becoming an increasingly prevalent method of casting a ballot. Switzerland and Canada began the practice sub-nationally in 2003, with Estonia offering it nationally starting in 2005 [7]. Adoption has grown steadily since. Over 500,000 municipal voters in Ontario (Canada) cast an online ballot in 2018 [1]. Almost 250,000 Estonians (representing 45% of participating voters) cast a ballot online in the 2019 Parliamentary elections.¹ And over 650,000 online voters participated in the 2021 State election in New South Wales (Australia).² Despite this rapid growth, few countries have developed adequate legislation or standards for online voting systems. In this under-regulated environment, online voting providers largely set their own security requirements, which has led to mixed outcomes.

¹ <https://valimised.ee/en/archive/statistics-about-internet-voting-estonia>.

² <https://elections.nsw.gov.au/About-us/Media-centre/News-media-releases/iVote-and-2021-NSW-Local-Government-elections>.

In this paper, we examine the question of ballot secrecy from the network perspective. Although ballot secrecy is a well-established requirement of democratic elections, the online voting setting offers new opportunities for exploitation. For example, suppose a network observer such as a internet service provider, content delivery network, or data center could determine how you voted. In that case, they could selectively prevent your ballot from reaching the election server to unduly influence the outcome of the election. Worse, with growing precedent for service disruptions and outages due to inadequate bandwidth³ on election night [1], a deliberate attack of this attack could escape detection.

Contributions. We present a novel ballot secrecy attack based on network traffic analysis of (encrypted) ballot confirmation pages. For a recent mayoral race in Canada, we demonstrate a classifier that could have correctly guessed voting intention for 84% of ballots based *only* on the byte-length of encrypted network traffic. Our results include:

- A detailed analysis of a real-world online voting system demonstrating the attack’s effectiveness in spite of well-configured TLS and variable-length HTML and DOM elements.
- A coordinated disclosure with the affected vendor resulting in them rolling out a mitigation.
- An analysis of the broader industry’s susceptibility to this attack and a discussion of mitigation options.

The rest of the paper is organized as follows: Sect. 2 presents background and related work. Section 3 recounts our efforts to reach out to vendors to seek demos to their voter interface. Section 4 describes the basics of the Simply Voting system. Section 5 describes our overall testing methodology, including technical details of our approach replicating Simply Voting’s server functionality and collecting network data. Section 6 presents the results of a simple (single contest) attack on ballot secrecy. Section 7 extends the experiment to more complex ballot configurations. Finally, Sect. 8 describes our coordinated disclosure with Simply Voting, their mitigation strategy, and the approaches of the other (responsive) vendors.

2 Background and Related Work

Ballot secrecy in online elections has been studied in the context of active attacks, such as subverting TLS [2, 8], exploiting implementation vulnerabilities [11, 13], or by unacknowledged privileged access [4]. Little related-work has evidently explored passive attacks that focus on the *lengths* of exchanged messages. One of the first articulations of this risk is a requirement due to Volkamer and Krimmer [12] (emphasis added):

³ <https://zdnet.com/article/no-surprise-nsw-ivote-fails-during-local-council-elections/>.

The e-voting system SHALL ensure neither the vote itself nor the number of chosen candidates (including an empty ballot), nor a spoiled vote (eg, by using the length of the protocol messages depending on the approach) can be deduced by reading transmitted voting protocol messages.

Clark and Essex [3] considered the possibility of a network observer being able to differentiate a voter’s selection based on the length of encrypted traffic sent to the election server by the voter’s browser. They found Dominion Voting Systems encoded candidate names explicitly in the cast vote object. For example, they observed a vote for *Meghan Agosta* was sent in an (encrypted) POST as {"ChoiceName": "Meghan Agosta"}. They speculated this approach could be susceptible to network-based length attacks, but did not conduct an analysis.

More recently, Specter et al. [10] explored this question in the context of the Voatz mobile voting app. Like the Dominion example, Voatz explicitly encoded the chosen candidate’s name, sending it to the server along with associated metadata in an HTTP POST. The authors observed a difference in the transmitted byte length of packets between a ballot cast for a candidate with a “short” name versus one with a “long” name.

However, our own experience examining online voting implementations has generally found cast ballot objects have a *fixed length*, with selections represented either as a code or ciphertext. This approach seemingly precludes length-based analysis—so we thought.

3 Research Question and Scope

Our study began with a hypothesis: Do ballot confirmation pages leak information about a voter’s selections? In particular, if the page was generated at the server-side and sent to the client immediately prior to casting, the TLS record byte-length may reveal information about the selected candidate.

Testing this hypothesis required access to a real-world online voting implementation. However, we were unaware of any vendor who maintained a publicly accessible demonstration that we could examine. The sole exception we observed was Simply Voting, a Montreal-based online voting vendor. Simply Voting mostly focuses on *non-governmental* elections (schools, companies, unions, political parties, etc.), however they did run the elections of 28 cities (accounting for over 300,000 voters) in the 2018 Ontario Municipal Election [1].

3.1 Vendor Demo Access Requests

As explained in subsequent sections, we were able to confirm our hypothesis on Simply Voting’s demo website. But what about the industry at large? Following our coordinated disclosure with Simply Voting, we decided to reach out to companies who had run (or were likely to run) a civic election in the near term.

We emailed each company identifying ourselves as cybersecurity researchers requesting a demonstration of the ballot casting experience. For each vendor, we

recorded whether they responded to our request, whether we were granted access to a demo, whether it was vulnerable to length-based analysis, and if so, what mitigation strategy was employed. We gave each vendor 30 d to respond. The results are shown in Table 1. The observed mitigations are discussed in Sect. 8.

Table 1. Vendor responses to our demo request and associated findings.

	Responsive	Access granted	Vulnerable	Mitigation strategy
Dominion	No	No	Unknown	Unknown
Intelivote	No	No	Unknown	Unknown
Neuvote	Yes	Yes–Private	No	Client-side generation
Scytl	Yes	No ^a	Unknown	Unknown
Simply Voting	Yes	Yes–Public	Mitigated	Random-length padding
SwissPost	Yes	Yes–Private	No	Client-side generation
Voatz	No	No	Unknown	Unknown

^aAgreed in principle, but access not granted by time of writing.

4 Description of Simply Voting’s System

This section describes Simply Voting’s process for casting ballots and evaluates the possibility of a length-based inference at different parts of this process.

4.1 Ballot Casting Process

Step 1: Logging In. The voter navigates to `demo.simplyvoting.com` and logs in with the given user ID and password. The user’s full name is then included in the HTML of the subsequent pages they access during the session.

Step 2: Submitting Choice of Candidates. The voter is presented with a single ballot page, which contains a set of offices (e.g., Mayor and City Councillor) and candidates. The voter selects which candidates they would like to vote *for*, and presses the **Continue** button. This submits a form containing the voter’s choices to the server represented as fixed-length codes.

Step 3: Confirmation. A confirmation page is sent to the voter from `demo.simplyvoting.com`. The served HTML content of this page contains the voter’s name, as well as the name of the voter’s choice of candidate. Note that static content, like images, stylesheets, and scripts, is served from a different domain, `static.simplyvoting.com`, with a different IP address.

Step 4: Review and Submission. The voter may choose to go back to the previous page and change their choices. If they do, they will again be presented with a confirmation page. If they are satisfied with their choices, the voter clicks the **Confirm** button, and their ballot is submitted to the server.

4.2 Potential Side-channel Attacks in the Ballot Casting Process

One opportunity for a length-based attack is when a voter’s selections are sent to the sever, as was observed in the Voatz system [10]. The names of the chosen candidate names were being POSTed to the server as explicit, uncompressed text. By contrast, Simply Voting’s system only POSTs fixed-length candidate IDs. For example, a vote for *Cassandra De Rolo* as Committee President is encoded in the HTTP request to the server as `ballot_579193[]=5724277`. Conversely, a vote for the opposing candidate, *Fernanda Rodriguez*, is represented by `ballot_579193[]=5724278`.

But what happens if the server returns a confirmation page containing the explicit names of the voter’s selections?

The values of some of the DOM elements are unknown to a network observer, while others can be predicted or deduced (see Table 2 for the full list).

We hypothesized that the length and value of the chosen candidate’s name had at least some effect on the size of the confirmation page and could leak information under certain conditions.

Table 2. Confirmation page DOM elements with varying values

Element	Example	Length	Predictable	Changes
CSRF token	c9590a...67652	Fixed	No	By session
Vote serial	e600de...9683b	Fixed	No	By session
Static resource version	84932	Fixed	Yes	Weekly ^a
Text time remaining	5 min and 0 s	Varies	Likely ^b	Every second
Integer time remaining	300	Varies	Likely ^b	Every second
Voter name	Taher Elgamal	Varies	Varies ^c	Every voter
Chosen candidate(s)	Linda Marlene Eales	Varies	–	By ballot

^aUsed by Simply Voting to periodically invalidate browser caches of their static resources. We sampled it every few days during the testing period.

^bAn observer could reasonably guess this by applying an offset to the time observed on their own confirmation page. However, off-by-one errors are possible: to make our approach as conservative as possible, we do not rely on knowing the time in our testing.

^cCould plausibly be known by ISP or network administrator, see Sect. 5.3.

5 Methodology

To test our hypothesis that a voter’s choice could correlate to the TLS record length of the ballot confirmation page, we needed to make a large volume of requests for confirmation pages and analyze the data transferred. Simply Voting’s public demo of their service allows us to observe what data is transmitted from their servers in a realistic election setting. However, making tens of thousands

of requests to their servers would place an undue burden on their resources and could trigger their network intrusion detection systems. Instead, we created our own server that replicates their confirmation page functionality. We also designed an application that could automatically make thousands of browser requests to this service and log the response for later analysis.

5.1 Testing a Length-Based Side-channel Attack

We created a testing system composed of two parts: a Client Application (to mimic a set of voters) and a Server Application (to mimic the online voting system). Each ballot “cast” in the experiments below corresponded to an actual HTTP request made over the internet between our local Client and cloud-based Server applications.

We designed our applications to simulate an election where a voter is eligible to vote for one or more offices (e.g., Mayor, Councillor, Deputy Mayor), and may cast a vote for no more than *one* candidate for each office. A voter casts a single *ticket*, a combination of candidates selected for each office. This is a common electoral system for municipalities in Ontario. Some Ontario municipalities use at-large systems,⁴ but this paper does not examine those elections.

5.2 Technical Implementation of the Client Application

We created the Client Application using Python, Selenium WebDriver, Google Chrome, and Wireshark. It was designed to make requests for confirmation pages, programmatically capture the response at the network layer, parse the TLS record length, and log the candidate choice and TLS record length to a file for statistical analysis. Our test bench is extensible and programmable: The client can decide which ballot to render by sending descriptive JSON to the server. The client can also set the flags to modify server behavior. For example, we implemented a flag that could programmatically enable/disable Simply Voting’s X-Ballot-Secrecy header (see Sect. 8.3).

The Client Application takes the following steps while interacting with the Server Application:

Table 3. 2018 municipal ballot options in Ward Ennismore, township of Selwyn

Mayoral candidate	Council candidate
Linda Marlene Eales	Donna Ballantyne
Andy Mitchell	Brad Sinclair
Ron Black	ABSTAIN
ABSTAIN	

⁴ <https://guelph.ca/wp-content/uploads/Ward-councillors-or-councillors-at-large.pdf>.

1. Client App is provided a list of offices and candidates (see e.g., Table 3).
2. Let o be the total number of offices and let $C_1, C_2 \dots C_o$ represent the set of choices available to a voter for each respective office (including abstain). The set of all possible candidate combinations (also known as *tickets*) that could be submitted by a voter T , is $(C_1)(C_2) \dots (C_o)$. The Client Application generates $|T|n$ tickets, where n is the required sample size for each ticket.
3. In its main process, the client requests a ballot confirmation page from the Server Application using Google Chrome automated with Selenium WebDriver. The confirmation page contains one ticket in T . The main process of the Client Application then listens to a message queue.
4. A second process (the *listening process*) uses Wireshark’s Python API⁵ to continuously listen to responses from the server application. When a response is detected, it records the TLS record length and pushes its value into the message queue.
5. The Client Application’s main process receives a TLS record length from the listening process in the message queue. Each observed record length (and the associated candidate) is appended to a CSV file. Steps 3 to 5 are repeated $|T|n$ times, until the test is complete.

5.3 Technical Implementation of the Server Application

Our goal was to replicate Simply Voting’s confirmation page functionality as faithfully as possible. To that end, we studied Simply Voting’s server stack and voting application by analyzing headers and interacting with their publicly accessible demo. We then matched this server stack as closely as possible, choosing popular and up-to-date software to fill gaps in the stack where Simply Voting’s choice was unknown (e.g., the server OS).

Observing Simply Voting’s Server Stack. We used several methods to learn about Simply Voting’s application configuration. We performed an SSL test⁶ to determine their supported and preferred encryption methods and analyzed the server headers sent to us while interacting with the demo application. We were able to determine the following relevant information about their server configuration:

- `demo.simplyvoting.com` reports its server software is Apache.
- The contents of the confirmation page are compressed via gzip.
- The confirmation page is streamed to the client with chunked transfer-encoding. However, in practice, only one chunk is transferred.⁷
- The TLS cipher suite on Windows and Linux desktops running Firefox or Chrome is `TLS_AES_256_GCM_SHA384`.⁸

⁵ <https://github.com/KimiNewt/pyshark/>.

⁶ <https://www.ssllabs.com/ssltest/>.

⁷ We tested chunked transfer-encoding on and found it made no significant difference in the ability to distinguish different ballots in our tests.

⁸ The chosen ciphersuite does not impact the feasibility of our attack. An observer can compute a separate record-length distribution for each observed ciphersuite.

Approximating Simply Voting’s Server Stack. We rented a Virtual Private Server (VPS) from ChunkHost to use as our replicated voting server, connected it to a domain name, and obtained a TLS certificate from Let’s Encrypt. We then deployed our Server Application with the following stack:

- **Debian 11.3 as the OS.** While we do not know what OS Simply Voting’s servers use, Debian is an operating system with considerable market share in the server space, and 11.3 was the latest release at the time of writing.
- **Apache 2.4.52 as the server.** Simply Voting reported in its headers that it used Apache, and Apache 2.4 was the most recent minor version.
- **Flask/Python 3.9 as the web framework.** Simply Voting’s web framework is unknown to us. For consistency with our client and analysis applications, we chose a Python-based web framework, and Flask is a mature Python web framework that met our relatively simple use case.
- **The TLS ciphersuite was forced to TLS_AES_256_GCM_SHA384.** This is the same as the TLS cipher suite preferred by Simply Voting on Windows and Linux desktops with major browsers.
- **Apache’s HTTP response headers** were manually overridden to match to Simply Voting’s.

Replicating Simply Voting’s Web Application. Our Server System re-implements Simply Voting’s ballot confirmation page. Upon receiving a request from the Client Application, the Server Application generates a confirmation page HTML document containing the data in Table 2, compresses it with GZIP, encrypts it with TLS_AES_256_GCM_SHA384, and serves it to the Client Application. Table 2 shows the elements with varying contents in the confirmation page, and our implementation substitutes appropriate values for all DOM elements with dynamic content:

- The Server Application generates random CSRF tokens and Vote Serials for each request.
- The Application assumes the Static Resource Version is fixed, as we observed it did not change for days at a time.
- The Server Application kept the voter’s name static across our trials for several reasons. First, real-world municipal elections do not include the voter’s name in the web session [1]. The voter’s name may be present in non-civic elections (unions, student clubs, and political parties). Even in these cases, two further reasons exist for assuming the voter’s name is known. First, the likely threat actors (e.g., internet service providers, family members, and cellular carriers) could plausibly associate a voter’s TLS session with their identity and compute a distribution of TLS record lengths for a voter with that name. Second, to meaningfully abuse ballot secrecy vulnerabilities in many cases, it is necessary to already know the identity of the voter whose ballot is being observed.⁹

⁹ In the case of a selective network outage attack, only the chosen candidate (not the voter’s name) is relevant to the attacker.


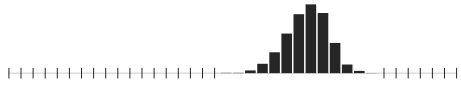


- The Application makes a conservative assumption that the time remaining varies within a 48- to 72-h window before voting closes. A more sophisticated observer may be able to increase the accuracy of their predictions by building a distribution with a more narrow time window to better approximate when a voter casts a ballot.
- The Server Application inserts the candidate choice that is requested by the Client Application.

6 Experiment 1 (Single Contest): Township of Selwyn, Ward Lakefield

6.1 Data Collection

In our first experiment, we replicated the behavior of a simple confirmation page offering a single choice for a single office, with a substantial length difference for each candidate name. One Ontario municipality that used Simply Voting during the 2018 municipal election meeting this criterion was the Township of Selwyn.¹⁰ In 2018, voters in Ward Lakefield were eligible to vote for a Mayor, Deputy Mayor, and a Councillor. However, the positions of Deputy Mayor and Councillor were uncontested, so voters only cast a ballot for Mayor. Voters had four possible choices: Linda Marlene Eales, Andrew Mitchell, Ron Black, and Abstain.

Table 4. Observed TLS record lengths (2,000 trials per candidate)

Candidate	Frequency of occurrence	Length (Bytes)		
		Min	Mean	Max
Abstain		3,301	3,306	3,311
Ron Black		3,319	3,326	3,331
Andy Mitchell		3,322	3,329	3,334
Linda Marlene Eales		3,327	3,333	3,338

Using our Client/Server test bench described in the previous section, we cast 2,000 ballots for each candidate: While we used the actual candidate names from this contest, we simulated an equal proportion of votes for each choice instead of

¹⁰ <https://elections.amo.on.ca/web/en/municipal/19401>.

the proportions of the actual election result. We recorded the TLS record length for each confirmation page returned by the Server Application. The distribution of TLS record lengths for each candidate choice is shown in Table 4.

6.2 Data Analysis

We want to estimate the probability that an encrypted vote V with byte length B is for candidate k , i.e., $\pi(V_k|B)$. To classify which candidate the encrypted vote is for a given byte length, we choose the candidate who maximizes the posterior probability:

$$\begin{aligned}\widehat{V}_k &= \arg \max_{k \in K} \{\pi(V_k|B)\} \\ &= \arg \max_{k \in K} \{\pi(B|V_k)\pi(V_k)\}.\end{aligned}$$

Generally, $\pi(B|V_k)$ is unknown. However, we can use simplifying assumptions to facilitate prediction. In particular, if we consider byte length as a categorical variable, then we can assume the likelihood for byte length is multinomial

$$\pi(B|V_k) = \text{Multinomial}(\boldsymbol{\theta}_k).$$

Here, the multinomial parameter $\boldsymbol{\theta}_k$ is indexed by k to allow for different candidates to have different probabilities for observing various byte lengths. Making this assumption on the likelihood leads to the *Multinomial Naive Bayes Model*. Using data with labelled votes and byte lengths, $\boldsymbol{\theta}_k$ can be estimated and then used to make predictions.

Using Python and `scikit-learn` [9], we ingest the data recorded by the Client Application and fit a Multinomial Naive Bayes Model and evaluate its out-of-sample performance on predicting which candidate a vote is for given the encrypted vote’s byte length. To estimate our model’s out-of-sample performance, we randomly split our data, using half to train the model and the other half to assess the accuracy of the model. The training set was used to fit our model. The performance metrics we present below are based on the predictions made on this test set. All data and code used in our analysis is available online.¹¹

We evaluate model classification ability using three metrics: accuracy, precision, and recall. The ballot in this example has four choices, and we simulated an equal proportion of results for each choice. This means that the best accuracy that should be achieved for a random guess—at least in theory—is 25%.

Result. The Naive Bayes model yielded an accuracy, precision, and recall on the test set of 83%, meaning 83 of every 100 votes from a simple random sample are correctly classified using byte length alone. Class-specific accuracy varies among candidates, with some candidates seeing very high accuracy (89%) while others see smaller accuracy (58%). However, accuracy across all classes is consistently larger than the expected 25%.

¹¹ <https://github.com/dpananos/ballot>.

True Label	Abstain	1	0	0	0
	Black	0	0.86	.13	.01
	Mitchell	0	0.26	0.58	0.16
	Eales	0	0	0.11	0.89
		Abstain	Black	Mitchell	Eales
		Predicted Label			

Fig. 1. Confusion Matrix (Proportions), Experiment 1. Rows normalized to sum to 1. Diagonal entries indicate class candidate-specific accuracy, while the other cells indicate proportion of votes for row candidate predicted to be the column candidate. As an example, 86% of votes for Black were correctly predicted to be for Black. 13% of votes for Black were predicted to be for Mitchell. The remaining 1% of votes for Black were predicted to be for Eales.

Figure 1, the confusion matrix, shows details about the predictions made by the Naive Bayes model on our test set. Voter choices are ordered by their mean TLS record length: It is apparent that the model is only confusing voter choices that are closest to each other in mean length. This property proves useful in later analyses of more complex elections. See *Identifying a Subset of Possible Candidate Combinations* in Sect. 7.1.

7 Additional Experiments

We conducted additional experiments with more complex confirmation pages that contain voter choices for multiple offices.

7.1 Experiment 2 (Two Contests): Township of Selwyn, Ward Ennismore

In 2018, voters in Ward Ennismore had four possible choices for mayor and three possible choices for Councillor, listed in Table 3. This results in twelve possible unique candidate combinations (tickets). We collected 500 samples per combination, for a total of 6,000 samples. Fitting a Multinomial Naive Bayes Model, we find values for accuracy, precision, and recall in Table 5. In general, performance is lower than in Experiment 1 because the length variation of different confirmation pages for the same candidate is greater. The variation increases due to

Table 5. Performance on test set by office, Experiment 2.

	Mayor			Councillor		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Naive Bayes	65%	75%	65%	50%	58%	51%
Random guessing	25%	25%	25%	33%	33%	33%

candidates for other offices being present on the confirmation page: they vary independently from the candidate being predicted.

Identifying a Subset of Possible Candidate Combinations. We also consider a more relaxed definition of violating ballot secrecy. Given a certain TLS record length, if we could identify a subset of possible candidate combinations that were chosen, that would also violate ballot secrecy. For each byte length, we counted the number of ballot configurations that produced record lengths of that byte length. Table 6 shows the proportion of ballots that have a TLS record length unique to a subset of possible candidate combinations.

Here, a possible candidate combination of n means that record length was sufficient to identify a vote to within n out of the 12 possible candidate combinations. Of note, 100% of ballots are associated with at most 11 possible candidate combinations, meaning that limited information about a voter’s choice is leaked for every ballot. In other words, for all ballots, we know at least one combination of candidates that were *not* chosen by the voter.

Table 6. Proportion of ballots by possible candidate combinations, Experiment 2 (Cumulative).

	Possible candidate combinations											
	1	2	3	4	5	6	7	8	9	10	11	12
Proportion	8%	11%	14%	19%	22%	25%	37%	43%	69%	90%	100%	100%

7.2 Experiment 3 (Three Contests): Town of Ajax, Ward 1

In 2018, voters in Ajax Ward 1 had six possible choices for Mayor, three possible choices for Regional Councillor, and seven possible choices for Councillor, resulting in 126 possible candidate combinations. We collected 987–1052 samples for each combination, for a total of 128,094 samples collected. Fitting a Multinomial Naive Bayes Model, we find values for accuracy, precision, and recall in Table 7. In general, performance is lower than in Experiments 1 and 2 because of even length variations introduced by a larger set of candidates for other offices.

Candidate Combination Subsets. By viewing the TLS record lengths of different candidate combinations, we show that we can still compromise ballot

secrecy (albeit to a limited extent) for all ballots in a manner similar to Experiment 2. Of the 126 possible candidate combinations (tickets), we found:

- 1% of all ballots had a unique TLS record length for that candidate combination
- 12% of all ballots cast had TLS record lengths that were shared with 10 or fewer other candidate combinations
- 53% of all ballots cast had TLS record lengths that were shared with 73 or fewer other candidate combinations
- 100% of all ballots cast had TLS record lengths that were shared with 92 or fewer other candidate combinations. In other words, for all votes cast in this election, we know at least 33 different ways to mark a ballot that was not chosen by the voter.

Table 7. Performance on test set by office, Experiment 3.

	Mayor			Councillor			Regional Councillor		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Bayes	33%	32%	33%	32%	33%	32%	63%	70%	63%
Guessing	17%	17%	17%	14%	14%	14%	33%	33%	33%

8 Mitigations

8.1 Client-Side Confirmation Page Generation

Transmitting the confirmation page over the internet can be avoided by generating the confirmation page on the client side in JavaScript. We observed the SwissPost and Neuvote systems taking this approach, rendering this particular side-channel *not-applicable*.

We met separately with representatives from Neuvote and Swiss Post and were granted private access to their (respective) demo systems. In both cases, we performed a basic analysis by casting ballots and observing the responses in Charles (an HTTP proxy) and Wireshark. We observed no ballot-related network activity in the time between selecting a candidate and rendering the confirmation page, indicating the page is generated on the client-side. We additionally observed that the cast ballot selections were encrypted at the application layer before being transmitted to the server. As expected, our experimental observations of packet lengths in Wireshark showed no perceptible correlation between candidate name length and network response length.

8.2 Fixed-Length Responses

Much discussion exists on the mitigation of length-based fingerprinting attacks, including adding padding to ensure the response is always of a fixed length. Gellert et al. describe such a scheme as “perfect length-hiding padding”, but also outline major performance tradeoffs [6].

We discussed this option with Simply Voting, but the practical limitations quickly became apparent. First, the padded size would need to be larger than the largest naturally-occurring response. The second is that the gzipped length is non-linearly dependent on the content itself, requiring the padding to either be calculated and applied *after* compression or for compression to be disabled.

Padding applied dynamically as a server header after compression is an atypical use case and would likely be difficult using standard server software. Disabling compression would needlessly slow page load times, which is highly problematic for an application involving large numbers of users making requests in a short window (i.e., election night). By default, many servers only compress MIME text/HTML. One solution might be to display candidate names as fixed-length images, although this would not, on its own, rule out the possibility it could lead to other distinguishing events.

8.3 Uniformly Random-Length Padding in Response Header

Coordinated Disclosure with Simply Voting. Once we had confirmed our hypothesis with the results of Experiment 1, we contacted Simply Voting to make the coordinated disclosure. They acknowledged our result, which we discussed in-depth in a meeting. Overall, we found the interaction positive and constructive and commend them for their commitment to the disclosure process.

Following internal discussions with the engineers, they eventually settled on a mitigation involving adding a random amount of padding bytes sampled uniformly in the interval $[0, 1000)$. The sever added this padding in a new `X-Ballot-Secrecy` response header, which is now live on their ballot confirmation pages.

Analysis of Simply Voting’s Fix. We implemented Simply Voting’s mitigation on our cloned server. We then re-ran Experiment 1 (see Sect. 6), which had 4 ballot options. With this mitigation enabled, our prediction strategy now had an accuracy of approximately 25%—reduced to (nearly) random guessing.

However, candidates with longer names become disproportionately distinguishable in instances where the `X-Ballot-Secrecy` header sampled close to the maximal length. For example, when a voter casts a ballot for Linda Marlene Eales (the choice that produces the largest ballot selection), if the `X-Ballot-Secrecy` header is near maximal (e.g., 998, 999, or 1000 bytes), it will produce a total TLS record length that is impossible to achieve with any other candidate choice. In that case, a passive observer would be able to identify that this voter cast a ballot for Linda with a high degree of certainty.

This phenomenon also exists when the ballot secrecy header is very close to its minimal length (e.g., 0 bytes), and a voter chooses to abstain (the choice produces the shortest ballot).

To quantify this, we can perform a similar analysis to the one we did in Experiments 2 and 3; we view the maximum and minimum TLS record lengths produced by each ballot choice and identify where these distributions do not overlap. If we observe a record length outside of the distribution of one of the ballot choices, we can deduce the ballot was *not* cast for that candidate. We conducted 8,000 trials per candidate for a total sample size of 32,000. We found:

- 0.25% of all ballots had a unique TLS record length for the candidate choice
- 0.38% of all ballots had TLS record lengths that were shared with 2 or fewer other candidate choices
- 1.18% of all ballots had TLS record lengths that were shared with 3 or fewer other candidate choices
- 98.83% of all ballots had TLS record lengths within the distribution of all other candidate choices

Simply Voting’s mitigation substantially lowers the risk of the attack presented in this paper. Although a practical fix under the circumstances, it still poses a risk to ballot secrecy for some voters in some cases. Client-side confirmation page generation, therefore, should remain the eventual goal.

8.4 Padding from a Gaussian Distribution

Degabriele [5] addresses the issue of overlapping uniform length distributions in the context of the CRIME/BREACH attack, where multiple observations of the same ciphertext with random padding by an attacker can be used to leak actual record lengths. The problem is similar to the limitations we identified with uniform padding in the ballot secrecy context: An attacker can observe the difference in the maximum and minimum of overlapping distributions. Degabriele proposes mitigating this by using a truncated Gaussian distribution, reducing the number of items at the tail end of the distribution. Future work should study the extent to which this approach reduces the number of clearly identifiable ballots.

8.5 Discussion and Conclusion

Using the network-observed TLS record length of the voter’s vote confirmation page, our model predicted the chosen candidate in a recent real-world mayoral contest with 83% accuracy relative to random guessing (which had 25% accuracy). In more complex ballots, our model still outperformed random guessing. However, for a large subset of ballots cast in an election, we could still obtain limited information in the form of certain combinations of candidates who were *not* voted for. Validation of our models shows this performance difference is unlikely to be explained by sampling variation.

Perhaps the biggest takeaway for us, however, was how difficult it was to obtain access to voter demos. If the security of a civic election is in the public interest, companies should not need long internal deliberations to respond to a request to see what a voter already sees. In this regard, we hope the industry will eventually follow Simply Voting's example and offer demos *pro forma*.

Acknowledgements. Thanks to Simply Voting, Swiss Post, and Neuvote for providing demo access. Thanks also to Jeremy Clark, Alex Halderman, Matthew Heuman, Brian Lack, Nicole Goodman, Philip Stark, and the anonymous reviewers for their valuable feedback. This work was supported by the National Science and Engineering Research Council of Canada's Discovery Grant program.

References

1. Cardillo, A., Akinyokun, N., Essex, A.: Online voting in Ontario municipal elections: a conflict of legal principles and technology? In: Krimmer, R., et al. (eds.) E-Vote-ID 2019. LNCS, vol. 11759, pp. 67–82. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30625-0_5
2. Cardillo, A., Essex, A.: The threat of SSL/TLS stripping to online voting. In: Krimmer, R., et al. (eds.) E-Vote-ID 2018. LNCS, vol. 11143, pp. 35–50. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00419-4_3
3. Clark, J., Essex, A.: Internet voting for persons with disabilities - security assessment of vendor proposals. City of Toronto FOI Request 2014-01543 (2014). <https://verifiedvoting.org/wp-content/uploads/2020/07/Canada-2014-01543-security-report.pdf>
4. Culnane, C., Eldridge, M., Essex, A., Teague, V.: Trust implications of DDoS protection in online elections. In: Krimmer, R., Volkamer, M., Braun Binder, N., Kersting, N., Pereira, O., Schürmann, C. (eds.) E-Vote-ID 2017. LNCS, vol. 10615, pp. 127–145. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68687-5_8
5. Degabriele, J.P.: Hiding the lengths of encrypted messages via Gaussian padding. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 1549–1565 (2021)
6. Gellert, K., Jager, T., Lyu, L., Neuschulden, T.: On fingerprinting attacks and length-hiding encryption. In: Galbraith, S.D. (ed.) CT-RSA 2022. LNCS, vol. 13161, pp. 345–369. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-95312-6_15
7. Germann, M., Serdült, U.: Internet voting and turnout: evidence from Switzerland. *Elect. Stud.* **47**, 1–12 (2017)
8. Halderman, J.A., Teague, V.: The New South Wales iVote system: security failures and verification flaws in a live online election. In: Haenni, R., Koenig, R.E., Wikström, D. (eds.) VOTELID 2015. LNCS, vol. 9269, pp. 35–53. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22270-7_3
9. Pedregosa, F., Varoquaux, G., Gramfort, A., et al.: Scikit-learn: machine learning in Python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011)
10. Specter, M.A., Koppel, J., Weitzner, D.: The ballot is busted before the blockchain: a security analysis of Voatz, the first internet voting application used in US. Federal elections. In: 29th USENIX Security Symposium (USENIX Security 2020), pp. 1535–1553 (2020)

11. Springall, D., et al.: Security analysis of the Estonian internet voting system. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 703–715. ACM (2014)
12. Volkamer, M., Krimmer, R.: Requirements and evaluation techniques for online-voting. In: 6th International EGOV Conference (Electronic Government), pp. 37–46 (2007)
13. Wolchok, S., Wustrow, E., Isabel, D., Halderman, J.A.: Financial cryptography, chap. Attacking the Washington, D.C. Internet Voting System, pp. 114–128 (2012)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

