



# UNIVERSITY OF PADOVA

DEPARTMENT OF MATHEMATICS "TULLIO LEVI-CIVITA"

*MASTER THESIS IN CYBERSECURITY*

## **SIDE-CHANNEL ANALYSIS OF ENCRYPTED ONLINE VOTING SYSTEMS: INFERRING VOTE CHOICES FROM NETWORK TIMING PATTERNS**

*SUPERVISOR*

PROF. MAURO CONTI  
UNIVERSITY OF PADOVA

*CO-SUPERVISOR*

ANASTASIIA BELOUSOVA  
FRANCESCO MARCHIORI  
UNIVERSITY OF PADOVA

*MASTER CANDIDATE*

MAXWELL ANTHONY RYAN

*STUDENT ID*

2122314

*ACADEMIC YEAR*

2025-2026







# Abstract

Online voting systems must guarantee ballot secrecy, ensuring that no observer can determine how an individual has voted. Although these systems employ cryptographic protections such as client-side encryption, the network traffic generated during vote submission may still leak information through timing side-channels. A passive adversary observing encrypted traffic between a voter and the voting server could potentially exploit differences in packet timing, sizes, and sequencing to infer ballot content. Despite the growing adoption of online voting across institutional and governmental elections, there has been limited systematic investigation into whether such timing leakage exists across different voting platforms and architectures.

This thesis presents a reusable, automated pipeline for detecting and quantifying timing-based side-channel leakage in online voting systems. The pipeline generates controlled elections with known ground-truth ballots, captures per-voter network traffic, extracts packet-level timing sequences from vote submission bursts, and evaluates the distinguishability of ballot choices using a bidirectional LSTM classifier. The methodology was applied uniformly to five platforms: Polyas, Helios, Belenios, CIVS, and Election Runner. Of these, only Polyas exhibited statistically significant leakage ( $AUC = 0.595$ ,  $p = 0.004$ ), traced to a  $\sim 15$  ms content-dependent delay in the server’s ballot confirmation packet. The remaining four platforms showed no detectable leakage, with platforms employing client-side homomorphic encryption proving particularly resistant. The pipeline is designed to be extensible to additional voting platforms and adaptable for integration into continuous testing workflows, enabling developers to detect timing leakage as voting systems evolve.



# Contents

|                                                                              |      |
|------------------------------------------------------------------------------|------|
| ABSTRACT                                                                     | v    |
| LIST OF FIGURES                                                              | x    |
| LIST OF TABLES                                                               | xiii |
| LISTING OF ACRONYMS                                                          | xv   |
| 1 INTRODUCTION                                                               | i    |
| 2 BACKGROUND AND RELATED WORK                                                | 5    |
| 2.1 Online Voting Systems . . . . .                                          | 5    |
| 2.2 Related Work . . . . .                                                   | 6    |
| 2.2.1 Timing Side-Channel Attacks on Cryptographic Implementations . . . . . | 6    |
| 2.2.2 Website Fingerprinting . . . . .                                       | 7    |
| 2.2.3 Encrypted Video and Media Identification . . . . .                     | 8    |
| 2.2.4 Traffic Analysis of Online Voting Systems . . . . .                    | 8    |
| 2.2.5 Deep Learning for Encrypted Traffic Classification . . . . .           | 9    |
| 2.2.6 Positioning of the Present Work . . . . .                              | 9    |
| 3 SYSTEM AND THREAT MODEL                                                    | ii   |
| 3.1 System Model . . . . .                                                   | ii   |
| 3.2 Threat Model . . . . .                                                   | 12   |
| 4 METHODOLOGY                                                                | 15   |
| 4.1 Overview . . . . .                                                       | 15   |
| 4.2 Controlled Election Generation . . . . .                                 | 16   |
| 4.2.1 Voter account creation . . . . .                                       | 16   |
| 4.2.2 Candidate Assignment . . . . .                                         | 16   |
| 4.2.3 Automated Vote Casting . . . . .                                       | 16   |
| 4.3 Network Traffic Capture . . . . .                                        | 17   |
| 4.4 Feature Extraction . . . . .                                             | 17   |
| 4.4.1 TLS Stream Selection . . . . .                                         | 17   |
| 4.4.2 Burst Detection . . . . .                                              | 17   |
| 4.4.3 Timing Features . . . . .                                              | 18   |
| 4.5 Classification . . . . .                                                 | 18   |

|       |                                            |    |
|-------|--------------------------------------------|----|
| 4.6   | Evaluation Strategy . . . . .              | 19 |
| 4.6.1 | Cross-Validation . . . . .                 | 19 |
| 4.6.2 | Evaluation Metrics . . . . .               | 19 |
| 4.6.3 | Statistical Significance Testing . . . . . | 20 |
| 4.7   | Platform Adoption . . . . .                | 21 |
| 5     | RESULTS AND EVALUATION                     | 23 |
| 5.1   | Experimental Overview . . . . .            | 23 |
| 5.2   | Polyas . . . . .                           | 24 |
| 5.2.1 | Experimental Conditions . . . . .          | 24 |
| 5.2.2 | Results . . . . .                          | 25 |
| 5.2.3 | Analysis . . . . .                         | 25 |
| 5.3   | Helios . . . . .                           | 30 |
| 5.3.1 | Experimental Conditions . . . . .          | 30 |
| 5.3.2 | Results . . . . .                          | 30 |
| 5.3.3 | Analysis . . . . .                         | 31 |
| 5.4   | Belenios . . . . .                         | 31 |
| 5.4.1 | Experimental Conditions . . . . .          | 31 |
| 5.4.2 | Results . . . . .                          | 32 |
| 5.4.3 | Analysis . . . . .                         | 32 |
| 5.5   | CIVS . . . . .                             | 33 |
| 5.5.1 | Experimental Conditions . . . . .          | 33 |
| 5.5.2 | Results . . . . .                          | 33 |
| 5.5.3 | Analysis . . . . .                         | 34 |
| 5.6   | Election Runner . . . . .                  | 34 |
| 5.6.1 | Experimental Conditions . . . . .          | 34 |
| 5.6.2 | Results . . . . .                          | 35 |
| 5.6.3 | Analysis . . . . .                         | 35 |
| 5.7   | Cross-Platform Comparison . . . . .        | 36 |
| 6     | DISCUSSION                                 | 39 |
| 6.1   | Classifier Behaviour Analysis . . . . .    | 39 |
| 6.2   | Practical Implications . . . . .           | 40 |
| 6.3   | Limitations . . . . .                      | 40 |
| 6.4   | Countermeasure Discussion . . . . .        | 42 |
| 7     | CONCLUSION                                 | 45 |
| 7.1   | Summary . . . . .                          | 45 |
| 7.2   | Contributions . . . . .                    | 46 |
| 7.3   | Future Work . . . . .                      | 46 |

|                 |    |
|-----------------|----|
| REFERENCES      | 49 |
| ACKNOWLEDGMENTS | 53 |



## Listing of figures

|     |                                                                                            |    |
|-----|--------------------------------------------------------------------------------------------|----|
| 3.1 | System and Threat Model . . . . .                                                          | 13 |
| 4.1 | Overview of the automated voting data generation and classification pipeline . . . . .     | 22 |
| 5.1 | Distribution analysis of the 318-byte S2C confirmation packet IAT on Polyas (1/2). . . . . | 27 |
| 5.2 | Distribution analysis of the 318-byte S2C confirmation packet IAT on Polyas (2/2). . . . . | 28 |



# Listing of tables

|     |                                                                                                                                  |    |
|-----|----------------------------------------------------------------------------------------------------------------------------------|----|
| 5.1 | Summary of experimental parameters per platform. . . . .                                                                         | 24 |
| 5.2 | Polyas: per-fold LSTM results (leave-one-election-out). . . . .                                                                  | 25 |
| 5.3 | Statistical analysis of the 318-byte S2C confirmation packet inter-arrival time<br>on Polyas ( $n_A = 235, n_B = 245$ ). . . . . | 29 |
| 5.4 | Helios: initial run with class imbalance (250 vs 207). . . . .                                                                   | 31 |
| 5.5 | Belenios: LSTM classifier results (10-fold CV, 458 voters). . . . .                                                              | 32 |
| 5.6 | CIVS: per-fold LSTM results (10-fold CV). . . . .                                                                                | 33 |
| 5.7 | Election Runner: summary statistics (leave-one-election-out, 24 folds). . . . .                                                  | 35 |
| 5.8 | Cross-platform comparison of LSTM classifier performance. . . . .                                                                | 36 |



# Listing of acronyms

|                      |                                                          |
|----------------------|----------------------------------------------------------|
| <b>VoIP</b> .....    | Voice over Internet Protocol                             |
| <b>ROC-AUC</b> ..... | Receiver Operating Characteristic - Area Under the Curve |
| <b>WF</b> .....      | Website Fingerprinting                                   |
| <b>LSTM</b> .....    | Long Short Term Memory                                   |
| <b>TLS</b> .....     | Transport Layer Security                                 |
| <b>PCAP</b> .....    | Packet Capture                                           |
| <b>TCP</b> .....     | Transmission Control Protocol                            |
| <b>C2S</b> .....     | Client-to-Server                                         |
| <b>S2C</b> .....     | Server-to-Client                                         |
| <b>CIVS</b> .....    | Condorcet Internet Voting Service                        |
| <b>IMAP</b> .....    | Internet Message Access Protocol                         |



# 1

## Introduction

Online voting systems have seen increasing adoption over the last two decades, both through expanded participation in existing systems and new deployments. For example, online voting participation in Estonia has grown from less than 2% [1] of voters in 2005 to more than 51% [2] of voters in the 2023 parliamentary elections, while online municipal voting in Ontario, Canada expanded to more than 200 municipalities by 2022 [3]. Online voting has also seen an increase in adoption outside the political sphere in academic institutions, unions, and companies. In Czechia, 18 of 26 of its public universities adopted remote online voting for internal elections [4]. In the United States, the Communications Workers of America (CWA), one of the country's largest unions, representing around 700,000 members, passed a resolution at its 2021 convention to conduct its national officer elections electronically, citing the inefficiency and hours-long queues involved in its previous in-person ballot processes [5]. Finally, in Australia, the company Computershare, which provides shareholder meeting services, states that the proportion of votes submitted digitally increased by 22% between 2018 and 2022, as reported in Computershare's 2023 report [6].

Central to any legitimate election, whether online or in person, is the principle of ballot secrecy. A voter must be able to cast a vote without the fear that their decision can be observed, inferred, or linked back to them; such a principle is outlined in Article 21(3) of the United Nations Universal Declaration of Human Rights: "The will of the people shall be the basis of the authority of government; this will shall be expressed in periodic and genuine elections which shall be by universal and equal suffrage and shall be held by secret vote or by equivalent free

voting procedures.” [7]. In traditional paper-based elections, ballot secrecy is enforced through physical mechanisms: private voting booths, sealed ballot boxes, and the anonymization of ballots during counting. Online voting replaces these physical guarantees with cryptographic ones. Online voting platforms rely on techniques such as client-side encryption, anonymization tokens, and cryptographic vote shuffling to ensure that ballots cannot be linked back to individual voters.

However, cryptographic protections of ballot content alone do not eliminate all vectors through which voting behavior may be inferred. Even when the payload of a network communication is encrypted, metadata is freely available to any passive observer on the network path between the voter and the server. Such metadata includes packet sizes, the timings of individual packets, the direction of transmission, and the sequence of network exchanges that constitute the voting session. The field of traffic analysis has repeatedly demonstrated that such metadata can be exploited to reveal detailed information about encrypted communications. In the past, researchers have used timing and size patterns to fingerprint encrypted web pages [8], identify users on the Tor Network [9], and infer the content of encrypted VoIP conversations [10] without decrypting a single byte of the payload.

Despite this breadth of work in traffic analysis, its application to online voting systems remains very sparse, representing a significant gap. If an attacker is positioned on the network, such as an internet service provider, a local network administrator, or a state-level adversary, they may be able to infer a voter’s ballot choice from the timing characteristics of their encrypted vote submission, violating ballot secrecy regardless of the strength of the underlying cryptography. This threat is not merely theoretical: Specter et al.’s security analysis of Voatz, a mobile internet voting platform used in U.S. federal elections, found that a passive network adversary is capable of learning a user’s secret ballot from encrypted traffic [11]. The consequences of such a vulnerability are particularly severe in contexts where voters face coercion, intimidation, or social pressure, since the possibility of ballot inference can have a chilling effect on the right to free and fair elections.

This thesis addresses the gap by presenting a systematic methodology for detecting and quantifying timing-based side-channel leakage in online voting systems. The focus is on building a reusable automated pipeline that can be applied across different voting systems to assess whether their network traffic patterns leak information about ballot content. The methodology generates controlled elections with known ground truths regarding ballot content, captures per-voter network traffic during vote submission, extracts timing features from the relevant network exchanges, and evaluates whether a machine learning classifier can distinguish between

ballot choices at a rate above random guessing. This approach is demonstrated in multiple voting systems, providing evidence of where such leakage exists and equally importantly evidence of where it does not. Of the five platforms evaluated, only Polyas exhibited statistically significant timing leakage ( $AUC = 0.595$ ,  $p = 0.004$ ).

The rest of the thesis is structured as follows: Chapter 2 provides the background and reviews related work, covering timing side-channel attacks, website fingerprinting, encrypted traffic classification, and prior traffic analysis of online voting systems. Chapter 3 defines the system and threat model, specifying the assumed network position of the adversary, their capabilities and the security properties under evaluation. Chapter 4 describes the methodology, including the automated election pipeline, traffic capture and feature extraction process, and the LSTM classifier architecture. Chapter 5 presents the results and evaluation for each of the platforms tested, along with cross-platform comparison and statistical analysis. Chapter 6 discusses the implications of the findings, the limitations of the experimental approach and potential countermeasures. Chapter 7 concludes the thesis and outlines directions for future work.



# 2

## Background and Related Work

This chapter provides the technical background necessary to understand the contributions of this thesis. It covers the architecture and cryptographic mechanisms of the online voting systems under study, the principles of network traffic analysis as applied to encrypted communications, and the existing literature on side-channel attacks relevant to online voting.

### 2.1 ONLINE VOTING SYSTEMS

Online voting systems, also referred to as internet voting systems, allow voters to cast ballots remotely through a web browser or a dedicated application. Unlike direct-recording electronic (DRE) machines used at physical polling stations, online voting systems transmit ballots over public networks and must therefore contend with a broader threat surface that includes network-level adversaries. The fundamental challenge is to provide ballot secrecy, ensuring that no party can determine how a voter voted, while also providing verifiability, ensuring that voters and observers can confirm that ballots were correctly recorded and counted.

Modern online voting systems address these challenges through cryptographic protocols. The voter's ballot is encrypted in the browser before transmission, so the plaintext vote is never sent out over the network. The encrypted ballot is submitted to the election server, which stores it without decryption. After the election closes, the results are combined either through homomorphic tallying, where encrypted votes are combined without individual decryption, or through mixnet-based shuffling, where ballots are decrypted only after their association with

individual voters has been severed. Both approaches aim to ensure that ballot secrecy is preserved throughout the election process.

## 2.2 RELATED WORK

The inference of private information from encrypted network communications through timing and traffic analysis has a rich history spanning cryptographic side channels, website fingerprinting, and application-level traffic classification. This section reviews the key developments in each of these areas and positions the present work within the broader landscape.

### 2.2.1 TIMING SIDE-CHANNEL ATTACKS ON CRYPTOGRAPHIC IMPLEMENTATIONS

The foundational work in timing-based side-channel attacks was presented by Kocher [12], who demonstrated that variations in the execution time of cryptographic operations could be exploited to recover secret keys from implementations of Diffie–Hellman, RSA, and DSS. This result established that even mathematically secure cryptographic algorithms could leak information through their physical or computational execution characteristics.

Brumley and Boneh [13] extended this paradigm to networked settings, demonstrating that RSA private keys could be extracted from an OpenSSL-based web server over a local network by measuring the time taken to respond to SSL handshake messages. Their work was significant because it showed that network latency and jitter did not completely obscure the timing signal; with sufficient measurements, statistical techniques could compensate for noise introduced by the network. This result directly led to the widespread adoption of RSA blinding as a default countermeasure in cryptographic libraries. Brumley and Taveri [14] later showed that similar timing attacks remained practical against elliptic curve implementations in OpenSSL, demonstrating that the problem was not limited to RSA.

These attacks target the cryptographic layer directly, exploiting implementation-specific timing variations in mathematical operations. The present work operates at a higher level of abstraction: rather than attacking the cryptographic primitives themselves, we analyze timing patterns in the application-level network traffic produced during an online voting session. The cryptographic operations within the voting platforms we study are assumed to be correctly implemented; the question is whether the overall system architecture produces observable timing differences that correlate with ballot content.

### 2.2.2 WEBSITE FINGERPRINTING

Website fingerprinting (WF) is the most extensively studied form of traffic analysis against encrypted communications. In a WF attack, a passive observer analyzes metadata from encrypted web traffic - primarily packet sizes, directions, and inter-arrival times - to determine which website a user is visiting, despite the use of encryption or anonymization tools such as Tor.

Early WF attacks relied on handcrafted features and traditional machine learning classifiers. Panchenko et al. [15] demonstrated effective fingerprinting of Tor traffic using support vector machines trained on statistical features derived from packet sequences. Subsequent work by Wang et al. [16] introduced the k-nearest-neighbour-based attack k-FP, which achieved high accuracy even against defended Tor traffic.

The application of deep learning to website fingerprinting marked a significant advance. Rimmer et al. [17] evaluated stacked denoising autoencoders, convolutional neural networks (CNNs), and long short-term memory (LSTM) networks for automated website fingerprinting, demonstrating that deep learning could match or exceed the performance of hand-engineered feature sets. The most influential work in this area is Deep Fingerprinting by Sirinam et al. [18], which employed a deep CNN architecture to achieve over 98% accuracy on undefended Tor traffic and over 90% accuracy against the WTF-PAD defence. This result demonstrated that even lightweight traffic padding schemes were insufficient to prevent identification by deep learning classifiers.

More recently, Bhat et al. [19] proposed Var-CNN, which incorporated both packet direction sequences and timing information to achieve data-efficient website fingerprinting, while Rahman et al. [20] demonstrated in Tik-Tok that packet timing alone could serve as a powerful feature for WF attacks, reinforcing the discriminative value of inter-arrival times.

The present work draws methodological inspiration from website fingerprinting - particularly the use of packet-level sequences as input to deep learning classifiers - but addresses a fundamentally different question. Website fingerprinting seeks to identify *which* service a user accesses, where different websites produce structurally distinct traffic patterns (different page sizes, numbers of resources, request sequences). In contrast, our attack targets *what content* a user submits within a single service, where both candidates produce traffic through the same application flow and the same encrypted protocol. The traffic differences, if they exist, are therefore far more subtle, arising from content-dependent processing rather than structurally different page loads.

### 2.2.3 ENCRYPTED VIDEO AND MEDIA IDENTIFICATION

A related line of work has demonstrated that the content of encrypted streaming media can be inferred from traffic metadata. Adaptive bitrate streaming protocols such as MPEG-DASH transmit video in variable-length segments whose sizes depend on the content being streamed. Schuster et al. [21] showed that an eavesdropper could identify which video a user was watching on Netflix, YouTube, and Amazon Video with high accuracy by analyzing the sequence of encrypted segment sizes. Reed and Klimkowski [22] extended this to audio content, demonstrating that encrypted VoIP calls could reveal the language being spoken based on packet timing and size distributions.

These attacks succeed because the content being transmitted directly determines the structure of the encrypted traffic: different videos produce different bitrate profiles, and different spoken languages produce different patterns of voice activity. The analogy to our setting is imperfect but instructive: if the voting server’s response depends in any way on the ballot content, for instance, through content-dependent validation, different ciphertext sizes, or variable server-side processing, then the encrypted traffic may similarly reveal information about the vote.

### 2.2.4 TRAFFIC ANALYSIS OF ONLINE VOTING SYSTEMS

The most directly relevant prior work is by Belousova et al. [23], who conducted inference attacks on encrypted online voting traffic against two real-world platforms: Eligo and POLYAS. Their analysis demonstrated that a passive network observer could determine whether a voter had participated in an election, whether their ballot was valid or spoiled, and in some configurations, additional metadata about the voting session. They reported classification accuracies of up to 99% for participation detection and ballot validity inference on Eligo, and validated the generalizability of their approach on POLYAS.

Their work shares our threat model, a passive eavesdropper with access to TLS-encrypted traffic and confirms that online voting platforms are vulnerable to traffic analysis. However, their focus was on *participation detection* (did the voter cast a ballot?) and *ballot validity classification* (was the ballot valid or spoiled?), rather than *ballot content inference* (which candidate did the voter select?). Ballot content inference is a different problem: while participation and validity produce structurally different traffic flows (e.g., a complete voting session versus an aborted one), different candidate choices traverse the same voting flow and produce superficially identical encrypted traffic.

The present work complements their findings by targeting this inference task across a broader set of platforms. Where their study focused on Eligo and POLYAS, we evaluate five platforms with diverse architectures including open-source systems based on ElGamal homomorphic encryption (Helios, Belenios), Condorcet-based systems (CIVS), and commercial platforms (POLYAS, Sequent). This comparative approach allows us to identify which architectural properties correlate with resistance to timing-based ballot inference.

### 2.2.5 DEEP LEARNING FOR ENCRYPTED TRAFFIC CLASSIFICATION

Beyond website fingerprinting, deep learning has been widely applied to the classification of encrypted network traffic for purposes including application identification, malware detection, and device fingerprinting. Aceto et al. [24] provided a comprehensive framework for deep-learning-based encrypted traffic classification, evaluating CNNs, LSTMs, and autoencoders on mobile application traffic. Their work highlighted that LSTM networks were particularly well suited to traffic classification tasks due to their ability to model temporal dependencies in packet sequences.

Lotfollahi et al. [25] proposed Deep Packet, which integrated CNN and autoencoder architectures for end-to-end encrypted traffic classification, achieving over 95% accuracy on application identification tasks without manual feature engineering. Similarly, Lopez-Martin et al. [26] demonstrated that recurrent neural networks could effectively classify encrypted network flows using only packet-level metadata features such as sizes, directions, and inter-arrival times.

These results inform our choice of classifier architecture. We employ a bidirectional LSTM trained on per-packet feature sequences (inter-arrival time, packet direction, frame length) extracted from the vote submission burst. This approach preserves temporal ordering information that is lost by traditional feature aggregation methods such as tsfresh, and has been shown in the traffic classification literature to be effective for learning from packet-level metadata.

### 2.2.6 POSITIONING OF THE PRESENT WORK

The present work sits at the intersection of several of these research threads. From timing side-channel attacks, we adopt the principle that encrypted systems may leak information through observable timing variations. From website fingerprinting, we borrow the methodology of training classifiers on packet-level traffic metadata. From the literature on online voting traffic analysis, we inherit the threat model and the specific application domain. Our contribution is

threefold: (1) we target ballot content inference rather than participation or validity detection; (2) we evaluate a broad set of platforms with differing cryptographic and architectural properties; and (3) we propose a reusable, automated pipeline for generating labeled voting traffic datasets and evaluating side-channel leakage across arbitrary online voting systems.

# 3

## System and Threat Model

This section describes both the System Model (3.1) and the Threat Model (3.2) under which the work is carried out.

### 3.1 SYSTEM MODEL

The system comprises three principal components: the voter's device, the voting server, and the network path between them, as shown in Figure 3.1.

The voter's device is a standard computer running a web browser. The client-side code of the voting platform executes within the browser, presenting the ballot, recording the voter's choice, encrypting the ballot, and transmitting the encrypted ballot to the server. The voter authenticates using credentials distributed via email prior to the election.

The voting server hosts the election, serves the voting interface to the browser, receives encrypted ballots, and then stores them for later tallying. Depending on the platform, the server may perform validation of the ballot upon receipt, such as verifying zero-knowledge proofs, checking voter eligibility, or logging the submission.

The network path between the voter and the server carries TLS-encrypted traffic. This traffic traverses multiple network segments including the voter's local network, their ISP's infrastructure, and the server's hosting infrastructure. At each point along this path, an observer can see the encrypted packets, including their sizes, timings, and direction, but cannot read their contents.

A typical voting session generates the following network exchanges: a TLS handshake to establish the encrypted connection, one or more request-response pairs to load the voting interface and authenticate the voter, and a final exchange to submit the encrypted ballot and receive confirmation. These exchanges manifest as bursts of packets separated by idle periods corresponding to voter interaction time (e.g. reading the ballot, making a decision).

## 3.2 THREAT MODEL

The attacker is a passive network observer positioned on the network path between the voter and the voting server, as shown in Figure 3.1. This represents adversaries such as an internet service provider, local network administrator (e.g. at a workplace or university), a compromised network appliance, or a state-level entity with access to backbone infrastructure.

The attacker can observe all encrypted packets exchanged between the voter's browser and the voting server. For each packet, the attacker can determine the timestamp, the size, and the direction of transmission (Client-to-Server or Server-to-Client). The attacker cannot decrypt the packet payload, as TLS encryption is assumed to be secure.

The attacker's goal is to infer the voter's ballot choice from the observable traffic metadata. Specifically, the attacker aims to determine which candidate the voter selected by analyzing timing patterns in the vote submission exchange.

The attacker has the following capabilities and knowledge. The attacker knows that an election is taking place and can identify traffic to the voting server by its destination IP address. The attacker knows the set of candidates in the election (which is typically public information). The attacker may have access to a training dataset of labeled traffic captures obtained by casting votes with known ballot choices on the same platform.

This threat model is conservative in the sense that a real-world attacker might have additional capabilities (e.g. active network manipulation) that could amplify timing signals. The results obtained under this passive model therefore represent a lower bound on information leakage available to a more capable adversary.

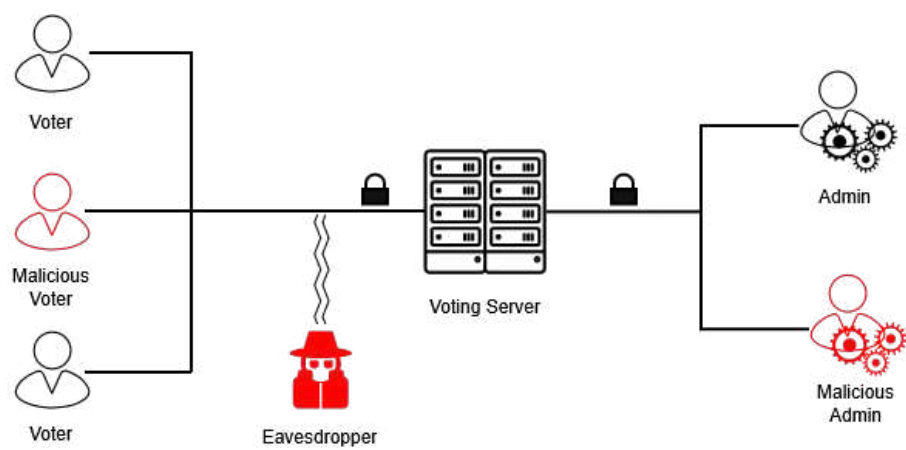


Figure 3.1: System and Threat Model



# 4

## Methodology

### 4.1 OVERVIEW

The core premise of the methodology is straightforward. If an online voting system leaks timing information about ballot content through its network traffic, then a classifier trained on labeled traffic captures should be able to distinguish between ballot choices at a rate above random guessing. A result at or near 50% accuracy (for a binary ballot) indicates that there is no detectable leakage, while a statistically significant result above 50% indicates that the encrypted network traffic carries information about the voters choice.

The pipeline operates as follows. First, a controlled election is created on the target platform with a known set of candidates. A pool of voter accounts are registered, and each voter is assigned a predetermined ballot choice, establishing ground truth labels. Second, an automated browsing script casts each vote sequentially while a packet capture tool records the network traffic for each individual voter. Third, the captured traffic is processed to isolate the vote submission exchange and extract timings from the relevant network packets. Finally, a machine learning classifier is trained and evaluated using cross-validation and the result is tested for statistical significance against the null hypothesis that accuracy equals chance.

This methodology is applied identically across platforms, with only the browser automation and burst separation timing threshold adapted to each platform. All other aspects of the pipeline remain constant, ensuring comparable results across platforms.

## 4.2 CONTROLLED ELECTION GENERATION

### 4.2.1 VOTER ACCOUNT CREATION

Voter accounts are created with the Gmail alias feature. A single email address serves as the base, with aliases generated by appending a voter index to the username (*e.g. baseaddress+voter1@gmail.com*). All aliases are delivered to the same inbox, allowing credentials to be easily retrieved programmatically. Each voting platform distributes voter credentials in the form of a voter ID and a one-time password via email to the registered address. The automation script connects to the Gmail inbox via IMAP, retrieves the most recent email addressed to each alias, and extracts the voting link, voter ID, and password using regular expressions matched to the platform’s email format.

### 4.2.2 CANDIDATE ASSIGNMENT

Each voter is assigned a predetermined candidate choice prior to the election. Candidates are evenly split to produce a balanced 50/50 class distribution. The assignment order is randomized and fixed with a deterministic seed, ensuring that votes for each candidate are interleaved throughout the recording session rather than grouped sequentially. This randomization is important, as without it, temporal effects, such as gradual changes in server load or network conditions over the duration of the recording session, could introduce confounding characteristics that correlate with the ballot label, producing spurious classifier accuracy.

### 4.2.3 AUTOMATED VOTE CASTING

Votes are cast using Playwright, a browser automation framework that drives a chromium instance through the voting process. A separate automation script is developed for each voting platform, mapping the platform’s specific user interface elements to a sequence of actions: navigating to the election URL, authenticating with the voter’s credentials, selecting the assigned candidate, confirming the ballot, and submitting the vote. Random delays are drawn from a Gaussian distribution (mean 1.5 seconds, standard deviation 0.3 seconds) are inserted between each user interaction to simulate realistic user behavior and avoid introducing artificially deterministic timings into the browser-side actions. The browser is launched with TLS session caching disabled to ensure that each voter establishes a fresh TLS connection, preventing session reuse from confounding the timing measurements.

## 4.3 NETWORK TRAFFIC CAPTURE

For each voter, a dedicated packet capture is initiated before the browser begins the voting process and is terminated after the vote is confirmed. Capture is performed using tcpdump, filtering for TCP traffic on port 443 to isolate TLS-encrypted communication with the voting server. Each capture is saved as an individual PCAP file with a filename encoding the voter number and the assigned candidate. This per-voter capture strategy ensures clean separation between voting sessions and allows each sample to be independently labeled and processed.

## 4.4 FEATURE EXTRACTION

The feature extraction stage processes each PCAP file to isolate the network exchange corresponding to the vote submission and extract timing features from the relevant packets. This stage is the most critical component of the pipeline, as the quality of the extracted features directly determines the classifiers ability to detect leakage.

### 4.4.1 TLS STREAM SELECTION

Each PCAP file contains traffic from multiple TLS streams, corresponding to the various connections established during the voting session (e.g connections to the voting server, content delivery networks, analytics services and other third-party resources). The vote-relevant stream is identified by selecting the TLS stream with the latest ending timestamp among those containing at least ten packets. This heuristic is based on the observation that vote submission is typically the final significant network exchange in the session.

### 4.4.2 BURST DETECTION

Within the selected TLS stream, packets are grouped into bursts based on inter-arrival time gaps. A burst boundary is defined where the time gap between consecutive packets exceeds a configurable threshold. The threshold is tuned per platform to reflect the characteristic timing of each system's network exchanges; for example, a threshold that cleanly separates the TLS handshake, page login, and vote submission on one platform may need adjustment for another platform with different server response characteristics. The final burst in the stream is identified as the vote submission burst, as the vote cast is the last significant interaction in the session. As

a robustness check, if the final burst contains fewer than three packets (indicating a straggler packet as opposed to a genuine exchange), the extraction falls back to the preceding burst.

Burst Detection produces a segmentation of the stream into discrete interaction phases. For a typical voting session, this yields an initial TLS handshake burst, one or more page interaction bursts, and a final vote submission burst. Only the vote submission burst is used for feature extraction, as it contains the network exchange that is most likely to be influenced by the ballot content.

#### 4.4.3 TIMING FEATURES

The vote submission burst is represented as an ordered sequence of per-packet features. Each packet in the burst is described by three values: the inter-arrival time in milliseconds from the previous packet (zero for the first packet), the direction of transmission encoded as a binary value (0 for Client-to-Server, 1 for Server-to-Client), and the frame length in bytes. This sequential representation preserves the temporal ordering of packets within the burst, allowing the LSTM to learn patterns in the sequence of timings and directions rather than relying on pre-computed summary statistics

### 4.5 CLASSIFICATION

A Long Short-Term Memory (LSTM) network is used to classify sequential packet-level features. The model consists of an input layer, a masking layer to handle variable-length sequences padded with zeros, one or more LSTM layers (default 32 units) with dropout, an optional dense layer (16 units with ReLU activation) and a Sigmoid output. The masking layer ensures that zero-padded timestamps do not contribute to the hidden state, allowing the model to process sequences of varying length without being confused by padding artifacts.

Input sequences are normalized by fitting a standard scaler on all timesteps of the training and testing set.

## 4.6 EVALUATION STRATEGY

### 4.6.1 CROSS-VALIDATION

Two cross-validation strategies are used, depending on the dataset structure. When the data set comprises multiple independently generated elections (as in the Polyas experiments, where ten separate elections of 49 voters each are conducted), leave-one-election-out cross validation is employed. In each fold, the model trains on data from nine elections and is tested on the held-out election. This strategy tests the model’s ability to generalize across elections conducted at different times under potentially different network conditions, providing a realistic assessment of how the classifier would perform on unseen election data.

When the dataset comprises a single election, stratified K-fold cross-validation is used with ten folds. The data is randomly split into ten groups of approximately equal size, with stratification ensuring each fold preserves the overall class distribution. The model trains on nine folds and tests on the remaining fold, rotating through all ten. The classifier automatically selects the appropriate strategy based on the number of distinct elections in the dataset.

### 4.6.2 EVALUATION METRICS

The classifier is evaluated using the following metrics, computed from the confusion matrix components: true positives ( $TP$ ), true negatives ( $TN$ ), false positives ( $FP$ ), and false negatives ( $FN$ ).

**ACCURACY** measures the proportion of all predictions that are correct:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.1)$$

While intuitive, accuracy can be misleading under class imbalance, as a model that predicts only the majority class can achieve deceptively high scores.

**PRECISION** measures the proportion of positive predictions that are correct:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (4.2)$$

A high precision indicates that when the classifier predicts a candidate, it is likely correct.

RECALL (SENSITIVITY) measures the proportion of actual positives that are correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (4.3)$$

A high recall indicates the classifier rarely misses true instances of the target class.

F1 SCORE is the harmonic mean of precision and recall, providing a single metric that balances both:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4.4)$$

The harmonic mean penalises extreme imbalances between precision and recall, making it more informative than accuracy when class distributions are uneven.

AUC-ROC is the Area Under the Receiver Operating Characteristic curve. The ROC curve plots the true positive rate ( $TPR$ ) against the false positive rate ( $FPR$ ) at varying classification thresholds:

$$TPR = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{FP + TN} \quad (4.5)$$

$$\text{AUC} = \int_0^1 TPR(FPR^{-1}(x)) \, dx \quad (4.6)$$

An AUC of 0.5 indicates performance equivalent to random guessing, while 1.0 indicates perfect discrimination. AUC is threshold-independent, making it a robust measure of a classifier's discriminative ability regardless of the decision boundary chosen.

### 4.6.3 STATISTICAL SIGNIFICANCE TESTING

Statistical Significance is assessed using a one-sample  $t$ -test under the null hypothesis  $H_0$ : the mean cross-validation accuracy equals 0.5 (chance level):

$$t = \frac{\bar{x} - 0.5}{s/\sqrt{k}} \quad (4.7)$$

where  $\bar{x}$  is the mean accuracy across  $k$  folds and  $s$  is the standard deviation. A  $p$ -value below 0.05 indicates that the observed accuracy is unlikely to have arisen by chance, providing evidence of genuine information leakage in the network traffic. The test is essential because modest improvements over chance could plausibly arise from noise in a small number of folds without statistical testing to confirm accuracy.

## 4.7 PLATFORM ADOPTION

The methodology is designed to be portable across voting platforms. Adapting the pipeline to a new platform requires only the development of a new browser automation script that maps the platform’s user interface to the vote casting sequence, and an adjustment to of the email parsing regular expressions to match the platform’s credential distribution format. The burst detection threshold may also require tuning to match the platform’s characteristic traffic patterns. All other components: the traffic capture process, feature extraction, classifiers, cross-validation framework, and statistical evaluation are reused without modification. This design supports the goal of a general-purpose leakage detection tool that can be applied systematically to evaluate ballot secrecy across the online voting ecosystem.

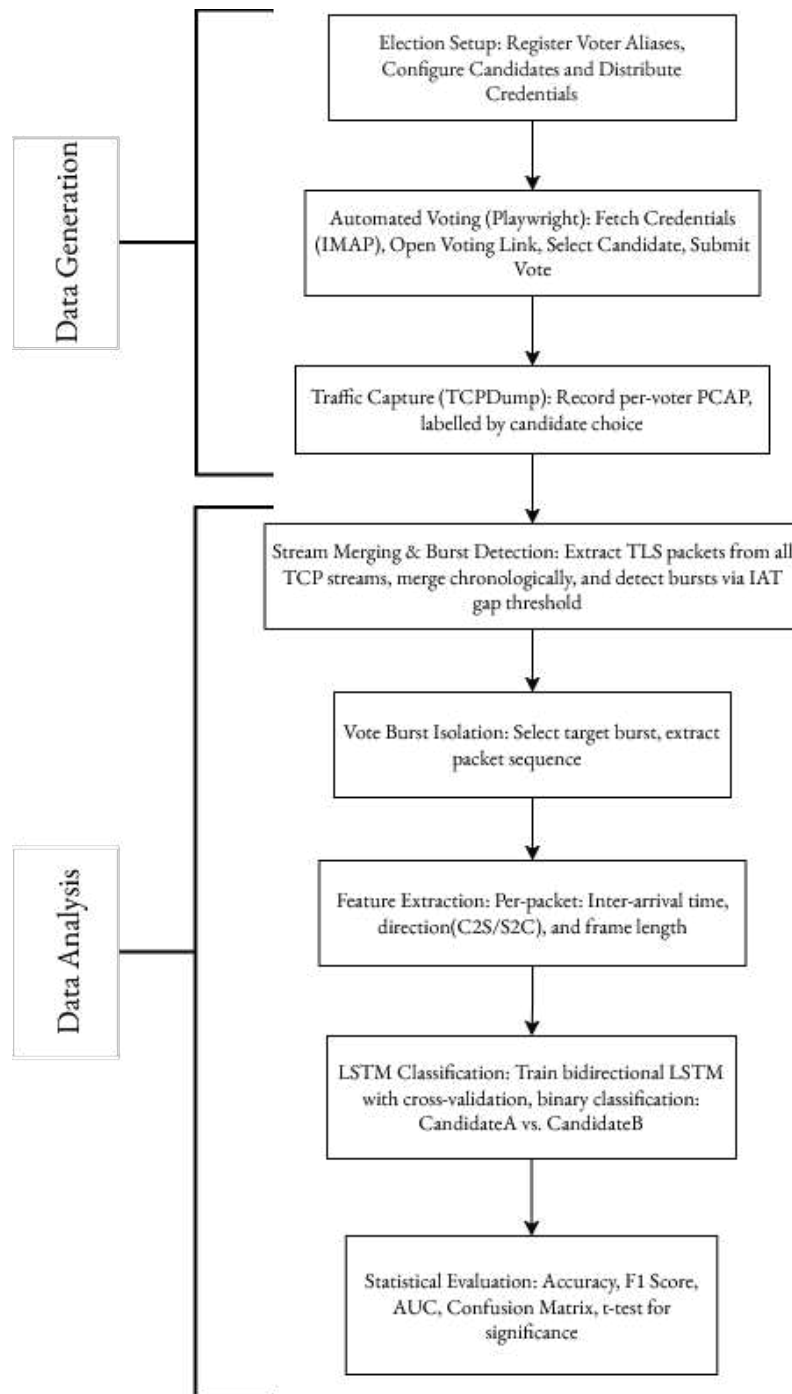


Figure 4.1: Overview of the automated voting data generation and classification pipeline

# 5

## Results and Evaluation

This chapter presents the experimental conditions, results and analysis for each voting platform evaluated using the methodology described in Chapter 4. For each platform, the specific election configuration and any platform-specific adaptations are described alongside the classifier results. The following is a cross-platform comparison and discussion of the findings.

### 5.1 EXPERIMENTAL OVERVIEW

All experiments share the common infrastructure described in the methodology: a VirtualBox virtual machine running Ubuntu 24.04 LTS, browser automation via Playwright, packet capture via tcpdump, and classification via a bidirectional LSTM trained on per-packet timing sequences (inter-arrival time, direction, frame length). Candidate assignment for all platforms follows a balanced 50/50 split with a fixed random seed for reproducibility, and candidates are randomly interleaved throughout each session to prevent temporal confounding.

Table 5.1 summarizes the experimental parameters for each platform.

**Table 5.1:** Summary of experimental parameters per platform.

| Platform        | Voters | Elections | CV Strategy   | Gap (s) | Stream Mode |
|-----------------|--------|-----------|---------------|---------|-------------|
| Polyas          | 490    | 10        | Leave-one-out | 1.0     | Single      |
| Helios          | 500    | 1         | 10-fold       | 0.5     | Single      |
| Belenios        | 458    | 1         | 10-fold       | 1.0     | Single      |
| CIVS            | 495    | 1         | 10-fold       | 1.0     | Multi       |
| Election Runner | 476    | 24        | Leave-one-out | 1.0     | Multi       |

## 5.2 POLYAS

### 5.2.1 EXPERIMENTAL CONDITIONS

Polyas is a closed-source commercial online voting platform widely used for organizational and institutional elections in Europe. Its cryptographic internals are not publicly documented, though it is certified under the Common Criteria Protection Profile for remote electronic voting systems.

A total of 490 voters were distributed across 10 separate elections of 49 voters each. Each election was created through the Polyas administration interface, and voter credentials were distributed via email to Gmail aliases. The two candidates used the same asymmetric naming scheme as all other platforms: CandidateA (10 characters) and Myxococcus llanfairpwll-gwyngyllgogerychwyrndrobwlllllantysiliogogochensis (73 characters), designed to maximize any potential timing signal from name-length-dependent processing. The vote burst for Polyas consisted of a compact exchange of 3–5 TLS packets: a session preparation packet, the encrypted ballot submission, a server acknowledgment, the cryptographic receipt, and a connection closure. A single-stream extraction mode was used, as the vote submission reliably occurred on a single TCP connection.

Six votes failed to register on the Polyas platform (the server did not record them), and four additional PCAPs had incomplete captures (missing the server confirmation packet). These 10 voters were excluded from the analysis, producing a clean data set of 480 voters. The failed votes were found to have systematically lower inter-arrival times, suggesting that the server rejected them quickly without performing full ballot processing, itself a noteworthy observation, as it indicates a passive observer could potentially distinguish failed from successful votes.

Cross-validation followed a leave-one-election-out strategy: for each fold, one election (49 voters) was held out for testing while the remaining nine elections (431 voters) were used for training. This ensures that the classifier is evaluated on an election it has never seen, testing

generalization across temporal and session-level variation.

### 5.2.2 RESULTS

**Table 5.2:** Polyas: per-fold LSTM results (leave-one-election-out).

| Fold (Test Election)                | Accuracy                            | AUC                                 | F1                                  |
|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Election 1                          | 0.571                               | 0.693                               | 0.618                               |
| Election 2                          | 0.469                               | 0.505                               | 0.409                               |
| Election 3                          | 0.612                               | 0.567                               | 0.698                               |
| Election 4                          | 0.571                               | 0.592                               | 0.512                               |
| Election 5                          | 0.510                               | 0.580                               | 0.539                               |
| Election 6                          | 0.571                               | 0.628                               | 0.432                               |
| Election 7                          | 0.531                               | 0.558                               | 0.489                               |
| Election 8                          | 0.653                               | 0.655                               | 0.702                               |
| Election 9                          | 0.571                               | 0.538                               | 0.588                               |
| Election 10                         | 0.653                               | 0.637                               | 0.691                               |
| <b>Mean <math>\pm</math> 95% CI</b> | <b>0.571 <math>\pm</math> 0.036</b> | <b>0.595 <math>\pm</math> 0.036</b> | <b>0.568 <math>\pm</math> 0.068</b> |

The aggregated confusion matrix across all folds is as follows.

|          | Predicted A | Predicted B |
|----------|-------------|-------------|
| Actual A | 134         | 106         |
| Actual B | 104         | 146         |

Sensitivity (recall for class B): 0.584. Specificity (recall for class A): 0.558. The one-sample  $t$ -test against chance (accuracy = 0.5) yields  $t = 3.86$ ,  $p = 0.004$ , indicating a statistically significant above-chance classification.

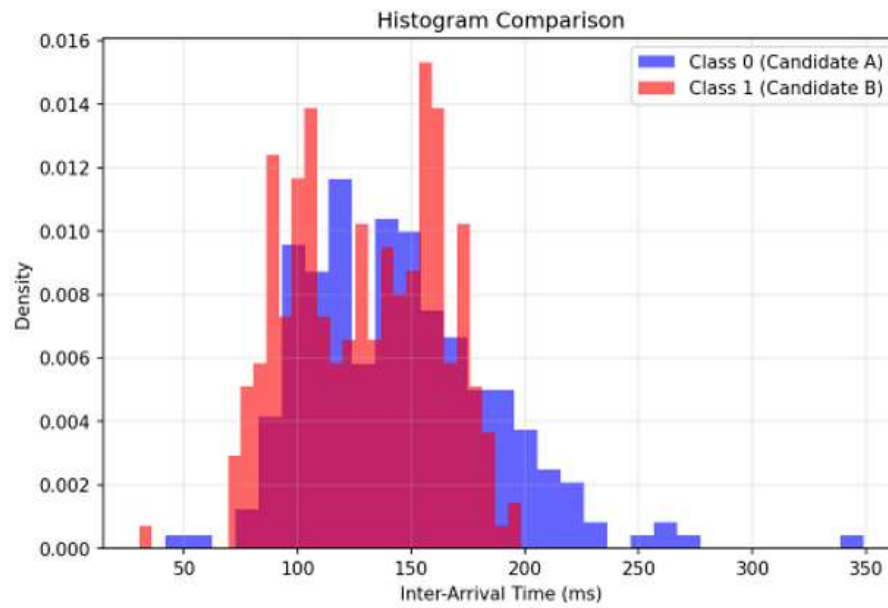
### 5.2.3 ANALYSIS

The classifier achieves a modest but statistically significant 57.1% accuracy and 59.5% AUC on Polyas. The confusion matrix shows balanced predictions (238 predicted A vs 252 predicted B), with both classes misclassified at similar rates ( $\sim 42\text{--}44\%$ ), confirming that the signal is weak but symmetric.

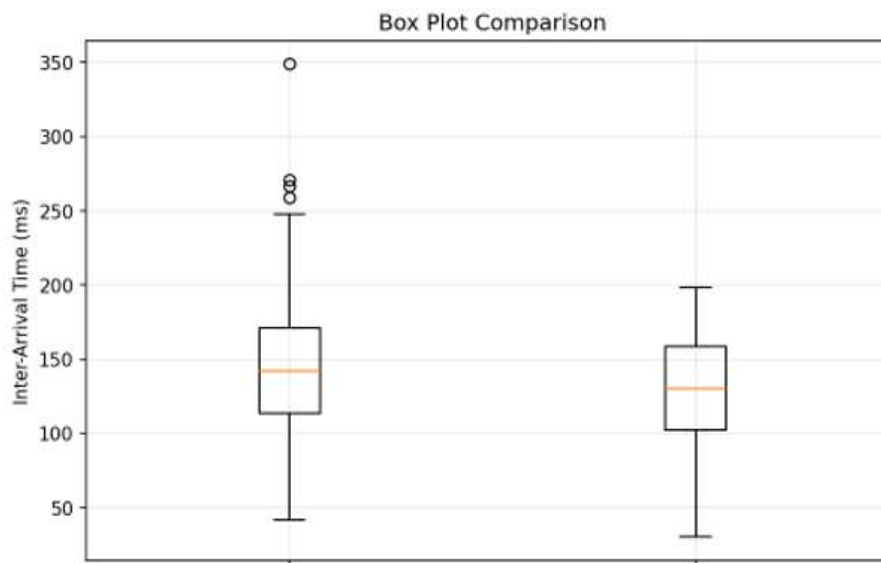
## PACKET-LEVEL LEAKAGE LOCALIZATION

To identify the source of the leakage, a detailed statistical analysis was conducted on each packet in the vote burst. The vote burst consists of a consistent 5-packet exchange: a session preparation packet (C2S, 184 bytes), encrypted ballot submission (C2S, 336 bytes), server acknowledgment (S2C, 95 bytes), cryptographic confirmation (S2C, 318 bytes), and connection closure (C2S, 95 bytes).

The analysis revealed that the 318-byte S2C confirmation packet is the main source of timing leakage. Figure 5.1 shows the distribution of inter-arrival times for this packet, separated by candidate class. Table 5.3 summarizes the key statistical findings.

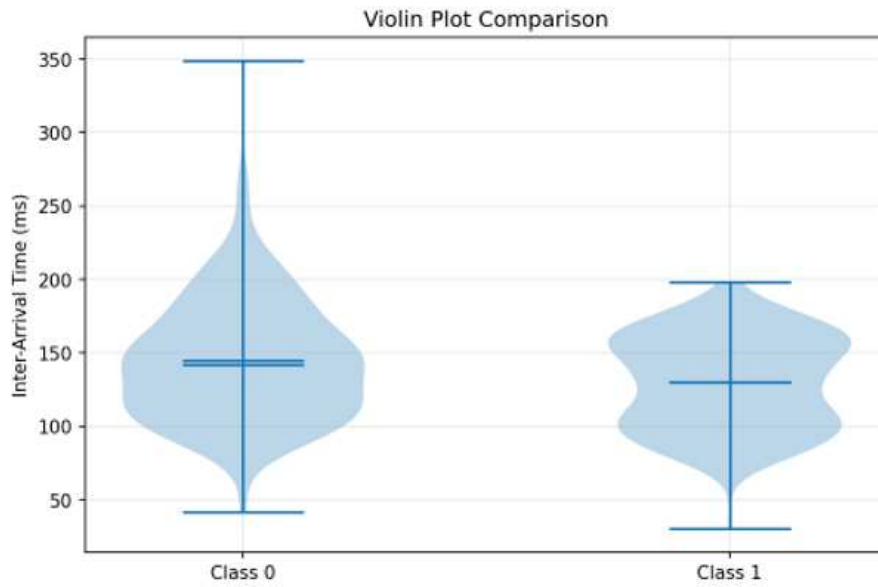


(a) Histogram comparison

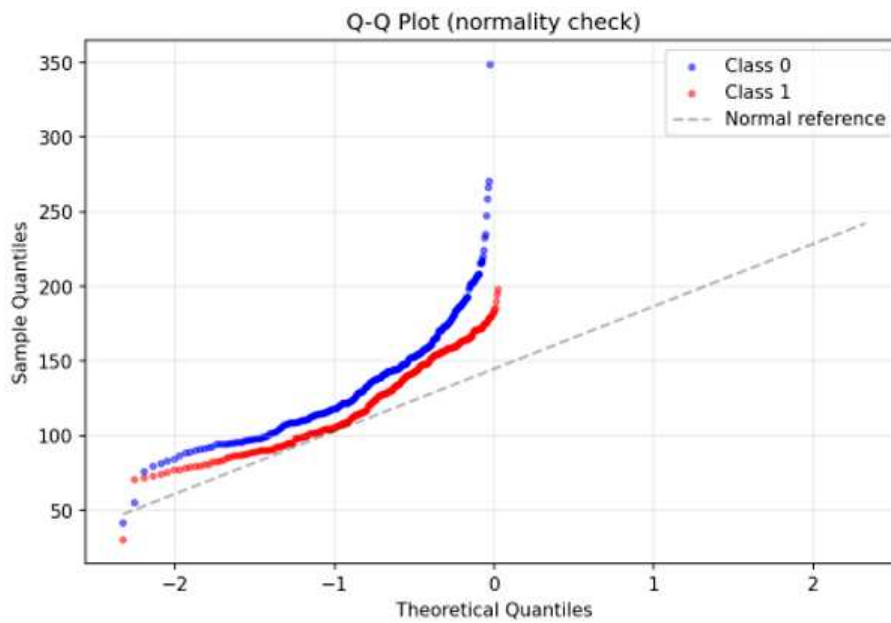


(b) Box plot comparison

Figure 5.1: Distribution analysis of the 318-byte S2C confirmation packet IAT on Polyas (1/2).



(a) Violin plot comparison



(b) Q-Q plot, confirms non-normality in both distributions

Figure 5.2: Distribution analysis of the 318-byte S2C confirmation packet IAT on Polyas (2/2).

The mean inter-arrival time for the 318-byte confirmation packet is 14.81 ms longer when CandidateA is selected compared to CandidateB. This difference is statistically significant across

**Table 5.3:** Statistical analysis of the 318-byte S2C confirmation packet inter-arrival time on Polyas ( $n_A = 235, n_B = 245$ ).

| Statistic                                                                         | CandidateA (Class 0) | CandidateB (Class 1) |
|-----------------------------------------------------------------------------------|----------------------|----------------------|
| Mean IAT (ms)                                                                     | 144.76               | 129.95               |
| Std. dev. (ms)                                                                    | 41.87                | 32.23                |
| Median IAT (ms)                                                                   | 141.90               | 130.25               |
| <b>Difference:</b> 14.81 ms (CandidateA slower)                                   |                      |                      |
| <b>Cohen's <math>d</math>:</b> $-0.396$                                           |                      |                      |
| <b>Overlap coefficient:</b> $0.560$                                               |                      |                      |
| <b>Mann–Whitney <math>U</math> <math>p</math>-value:</b> $< 0.001$                |                      |                      |
| <b>Welch's <math>t</math>-test <math>p</math>-value:</b> $< 0.001$ ( $t = 4.34$ ) |                      |                      |
| <b>Kolmogorov–Smirnov <math>p</math>-value:</b> $0.002$                           |                      |                      |

multiple tests: the Welch's  $t$ -test ( $p < 0.001$ ), the non-parametric Mann–Whitney  $U$  test ( $p < 0.001$ ), and the Kolmogorov–Smirnov two-sample test ( $p = 0.002$ ). Both distributions fail the Shapiro–Wilk normality test ( $p < 0.001$ ), justifying the use of non-parametric tests alongside the parametric ones.

The effect size (Cohen's  $d = -0.40$ ) falls in the small-to-medium range, and the distributional overlap coefficient of  $0.56$  indicates that  $56\%$  of the two distributions overlap. This heavy overlap explains why the classifier achieves only  $\sim 57\%$  accuracy despite the highly significant distributional difference: the signal exists in the aggregate but is too noisy for reliable individual-level inference.

#### INTERPRETATION

The 318-byte packet represents the server's cryptographic confirmation of the vote. The timing difference indicates that the server takes approximately 15 ms longer to generate this confirmation when CandidateA is selected. This is consistent with content-dependent server-side processing, the server's handling of the encrypted ballot differs depending on its content, even though the ballot is encrypted before submission.

CandidateA (CandidateA, 10 characters) produces a slower server response than CandidateB(Myxococcus llanfairpwllgwyngyllgogerychwyrndrobwlantysiliogogogochensis, 73 characters), which is counterintuitive if the length of the name were the cause. This suggests that the leak originates from a processing path that is not related to the length of the candidate name, possibly involving database operations, cryptographic receipt generation, or ballot validation logic within the Polyas server.

Feature importance analysis from earlier experiments with traditional classifiers (LightGBM)

on the same data confirmed that the top discriminative features were inter-arrival time quantiles (`iat_all__quantile__q_0.75`, `iat_all__maximum`, `iat_all__quantile__q_0.25`), reinforcing that the leakage resides in the *distribution shape* of packet timing rather than in a simple mean difference.

The practical implication is that Polyas exhibits a measurable timing side channel, but the effect size (Cohen’s  $d \approx 0.4$ ) is insufficient for reliable individual-level ballot inference. An adversary could gain some information about a voter’s choice, but could not determine it with certainty. However, as discussed in Section 6.2, even modest per-voter accuracy can become more exploitable when combined with prior knowledge of the overall vote distribution.

## 5.3 HELIOS

### 5.3.1 EXPERIMENTAL CONDITIONS

Helios is an open-source, web-based voting system that employs ElGamal homomorphic encryption. The ballot is encrypted entirely in the browser using JavaScript before submission; the server receives a uniform ciphertext regardless of the candidate selected. Helios was specifically chosen as a contrast to Polyas, as its client-side encryption architecture should theoretically produce no content-dependent server-side processing variation.

A single election of 500 voters was created on the Helios public server. To maximize any potential timing signal from candidate name length, the two candidates were given deliberately asymmetric names: CandidateA (10 characters) and Myxococcus *llanfairpwllgwyngyllgogerychwyrndrobwlantysiliogogochensis* (73 characters). If name length influenced ciphertext generation time, server-side validation, or any other processing step, this extreme asymmetry would amplify the effect. Of the 500 voters, 42 failed during automation (predominantly CandidateB voters), yielding an initial dataset of 458 voters with a class imbalance of 250 vs 207. The election was subsequently repeated with a balanced dataset.

The vote burst consisted of 4–6 TLS packets, and cross-validation used stratified 10-fold splits since all voters belonged to a single election.

### 5.3.2 RESULTS

On the balanced rerun: accuracy = 52.0%, AUC = 0.546,  $p = 0.40$  (not significant).

The initial imbalanced run (458 voters) showed superficially higher numbers, but with severely biased predictions:

**Table 5.4:** Helios: initial run with class imbalance (250 vs 207).

| <b>Metric</b> | <b>Mean</b> | <b><math>\pm</math> 95% CI</b> | <b><i>p</i>-value</b> |
|---------------|-------------|--------------------------------|-----------------------|
| Accuracy      | 0.552       | $\pm 0.056$                    | 0.106                 |
| AUC           | 0.576       | $\pm 0.072$                    | —                     |
| F1            | 0.372       | $\pm 0.098$                    | —                     |

The aggregated confusion matrix from the imbalanced run revealed that the model was overwhelmingly predicting class 0 (329 predictions of class 0 vs 128 of class 1), with a sensitivity of only 0.314 but a specificity of 0.748. The very low F1 score (0.372) confirmed that the model was barely detecting class 1. Once the class imbalance was corrected through a balanced rerun, the result dropped to chance level.

### 5.3.3 ANALYSIS

Helios shows no detectable timing leakage. The classifier performs at chance level ( $p = 0.40$ ), and this result holds despite the deliberately extreme candidate name asymmetry. This is consistent with Helios’s architecture: the ballot is encrypted client-side using ElGamal, producing a uniform-length ciphertext that the server processes identically regardless of content. The server never observes the choice of the plaintext ballot during the submission flow, and homomorphic encryption ensures that the ciphertexts of different candidates are computationally indistinguishable.

The initial false positive from the imbalanced run serves as a methodological lesson: class imbalance can create the appearance of above-chance performance through biased predictions, and must be carefully controlled for in any evaluation of this type.

## 5.4 BELENIOS

### 5.4.1 EXPERIMENTAL CONDITIONS

Belenios is an open-source voting system that, like Helios, employs ElGamal homomorphic encryption with client-side ballot encryption. It additionally features a two-phase authentication process: voters first receive a credential email with a unique voting link, then during the vote

submission process, a separate authentication code is sent to their email, which must be entered in a popup window before the ballot is cast.

A single election of 500 voters was created. Of these, 42 failed during the automation process, yielding 458 usable voters. The two candidates used the same asymmetric naming scheme as Helios (CandidateA vs the long Myxococcus name). The vote burst for Belenios was notably longer than the other platforms, consisting of 29–47 packets (mean 38.8) per voter, reflecting the more complex multi-step authentication and encryption protocol. Cross-validation used stratified 10-fold splits.

### 5.4.2 RESULTS

**Table 5.5:** Belenios: LSTM classifier results (10-fold CV, 458 voters).

| Metric   | Value | <i>p</i> -value | Significant? |
|----------|-------|-----------------|--------------|
| Accuracy | 0.472 | 0.13            | No           |
| AUC      | 0.475 |                 |              |

The model performs *below* chance, which is consistent with random guessing subject to noise. The train balanced accuracies across folds were very low (0.54–0.63), indicating that the model could not even learn from the data it was trained on.

### 5.4.3 ANALYSIS

Belenios does not show detectable leakage. This result is particularly noteworthy because the vote burst sequences were substantially longer (mean 38.8 packets) than those of Helios (4–6 packets) or Polyas (3–5 packets). If the classifier’s failure on Helios were attributable to insufficient data per sequence, one would expect better performance on Belenios, where the LSTM has far more temporal information to work with. The fact that accuracy remains at chance even with 7–10 times more packets per vote confirms that the absence of leakage is a property of the cryptographic design, not a limitation of the classifier or the amount of data available per vote.

The low train balanced accuracy further supports this interpretation: the model cannot learn any discriminative pattern even on the training data, ruling out the possibility that the model is simply too weak or underfit. The same LSTM architecture successfully detects leakage on Polyas, demonstrating that it is capable of finding timing signals when they exist.

## 5.5 CIVS

### 5.5.1 EXPERIMENTAL CONDITIONS

CIVS (Condorcet Internet Voting Service) is an open-source voting platform hosted by Cornell University that implements Condorcet-method ranked voting. Unlike the other platforms evaluated, CIVS requires voters to rank candidates by preference rather than selecting a single choice. For the purposes of this binary evaluation, voters ranked CandidateA first and CandidateB second, or vice versa. As with all other platforms, the two candidates used the asymmetric naming scheme: CandidateA (10 characters) and Myxococcus `llanfairpwllgwyngyllgongerychwyrndrobwlantysiliogogogochensis` (73 characters).

Each voter received a unique voting URL containing both an election identifier and a per-voter key. CIVS required email activation for each voter address before voting credentials could be issued, necessitating an additional automated activation step. A total of 495 voters were successfully processed. The vote burst consisted of a compact exchange following the submission of the ranking via an HTTP POST. Cross-validation used stratified 10-fold splits.

### 5.5.2 RESULTS

**Table 5.6:** CIVS: per-fold LSTM results (10-fold CV).

| Fold                                | Accuracy                            | AUC                                 | F1                                  |
|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Fold 1                              | 0.420                               | 0.317                               | 0.491                               |
| Fold 2                              | 0.420                               | 0.437                               | 0.383                               |
| Fold 3                              | 0.480                               | 0.438                               | 0.606                               |
| Fold 4                              | 0.600                               | 0.581                               | 0.615                               |
| Fold 5                              | 0.580                               | 0.621                               | 0.533                               |
| Fold 6                              | 0.388                               | 0.438                               | 0.348                               |
| Fold 7                              | 0.347                               | 0.367                               | 0.360                               |
| Fold 8                              | 0.469                               | 0.420                               | 0.552                               |
| Fold 9                              | 0.490                               | 0.517                               | 0.546                               |
| Fold 10                             | 0.510                               | 0.477                               | 0.333                               |
| <b>Mean <math>\pm</math> 95% CI</b> | <b>0.470 <math>\pm</math> 0.050</b> | <b>0.461 <math>\pm</math> 0.057</b> | <b>0.477 <math>\pm</math> 0.068</b> |

The aggregated confusion matrix across all folds is:  
Sensitivity: 0.504. Specificity: 0.437. The one-sample  $t$ -test yields  $t = -1.17$ ,  $p = 0.27$  (not significant). The classifier performs below chance, consistent with random guessing.

|          | Predicted A | Predicted B |
|----------|-------------|-------------|
| Actual A | 108         | 139         |
| Actual B | 123         | 125         |

### 5.5.3 ANALYSIS

CIVS shows no detectable timing leakage. The mean accuracy of 47.0% is below the 50% baseline, and the high variance across folds (ranging from 34.7% to 60.0%) is characteristic of a classifier that finds no consistent signal and produces noisy predictions. The confusion matrix shows a slight bias toward predicting class B (264 predictions of B vs 231 of A), but this does not produce above-chance accuracy.

CIVS differs architecturally from the other platforms in that its voting interface is a simple HTML form with dropdown selectors, submitted via a standard HTTP POST. The server-side processing is minimal, and the encrypted communication consists of a single request-response exchange. The absence of leakage is consistent with the simplicity of this interaction, there is little opportunity for content-dependent processing variation when the server receives a small, fixed-format ranking submission.

## 5.6 ELECTION RUNNER

### 5.6.1 EXPERIMENTAL CONDITIONS

Election Runner is a commercial online voting platform designed for organizational elections. It imposes a limit of 20 voters per election, requiring 25 separate elections to reach the target of 500 voters. Each election was created independently with the same two candidates using the asymmetric naming scheme common to all platforms (CandidateA vs the long *Myxococcus* name), and voter credentials were distributed via email. The same 20 Gmail aliases were reused throughout the elections, and old voting emails were deleted between runs to allow the IMAP-based credential retrieval to function correctly. Due to failures during automation, 24 elections were completed with a total of 476 voters (24 elections of 20 voters plus one partial election of 16).

The voting URL was embedded as an HTML hyperlink in the credential email rather than appearing in the plain text body, requiring HTML parsing for extraction. The vote submission involved selecting a candidate via a radio-button-style interface and clicking a “Submit Ballot” button. Cross-validation used a leave-one-election-out strategy across the 24 elections.

### 5.6.2 RESULTS

**Table 5.7:** Election Runner: summary statistics (leave-one-election-out, 24 folds).

| Metric   | Mean  | $\pm$ 95% CI | Min   | Max   |
|----------|-------|--------------|-------|-------|
| Accuracy | 0.574 | $\pm 0.036$  | 0.400 | 0.700 |
| AUC      | 0.519 | $\pm 0.044$  | 0.340 | 0.737 |
| F1       | 0.555 | $\pm 0.104$  | 0.000 | 0.800 |

The aggregated confusion matrix across all 24 folds is:

|          | Predicted A | Predicted B |
|----------|-------------|-------------|
| Actual A | 97          | 139         |
| Actual B | 64          | 176         |

Sensitivity: 0.733. Specificity: 0.411. The one-sample  $t$ -test on accuracy yields  $t = 4.02$ ,  $p = 0.0005$ , which is statistically significant.

### 5.6.3 ANALYSIS

Election Runner presents an ambiguous result that requires careful interpretation. While the accuracy of 57.4% is statistically significant ( $p < 0.001$ ), the AUC of 51.9% is essentially at chance. This disconnect arises because the classifier exhibits a strong prediction bias toward class B: 315 of 476 predictions (66%) are class B, despite the true distribution being nearly balanced (236 vs 240). The model achieves above-chance accuracy by over-predicting the slightly larger class, not by genuinely discriminating between candidates.

The AUC, being threshold-independent, provides a more honest assessment of discriminative ability. An AUC of 0.519 indicates that the model’s probabilistic rankings are barely better than random, even though the biased decision threshold happens to produce a statistically significant accuracy figure. The F1 scores reinforce this interpretation: three folds produced an F1 of 0.000 (the model predicted only one class for those elections), and the high variance ( $\pm 0.104$ ) reflects inconsistent behavior across folds.

The train balanced accuracies were uniformly low (0.56–0.58), indicating the model could not learn meaningful patterns even from the training data. This is consistent with the Helios, Belenios, and CIVS results. The small fold size (20 voters per election) also contributes to high variance and makes the per-fold results unreliable individually.

Based on the AUC and the prediction bias analysis, Election Runner is classified as showing **no reliable leakage**. The significant  $t$ -test on accuracy is an artifact of prediction bias rather than evidence of genuine ballot content inference.

## 5.7 CROSS-PLATFORM COMPARISON

Table 5.8 summarizes the classifier results across all evaluated platforms.

**Table 5.8:** Cross-platform comparison of LSTM classifier performance.

| Platform        | Accuracy          | AUC               | F1                | $p$ -value   | Leakage         |
|-----------------|-------------------|-------------------|-------------------|--------------|-----------------|
| Polyas          | $0.571 \pm 0.036$ | $0.595 \pm 0.036$ | $0.568 \pm 0.068$ | $0.004^{**}$ | Yes             |
| Helios          | 0.520             | 0.546             | —                 | 0.400        | No              |
| Belenios        | 0.472             | 0.475             | —                 | 0.130        | No              |
| CIVS            | $0.470 \pm 0.050$ | $0.461 \pm 0.057$ | $0.477 \pm 0.068$ | 0.274        | No              |
| Election Runner | $0.574 \pm 0.036$ | $0.519 \pm 0.044$ | $0.555 \pm 0.104$ | $0.001^*$    | No <sup>†</sup> |

<sup>\*\*</sup>  $p < 0.01$ . <sup>\*</sup> Significant on accuracy but AUC at chance; prediction bias suspected. <sup>†</sup> Accuracy significant but AUC (0.519) indicates no genuine discriminative ability.

A clear pattern emerges from the five platforms with completed results. Polyas, a commercial closed-source system, exhibits statistically significant timing leakage ( $p = 0.004$ ) with a meaningful AUC of 0.595. Helios, Belenios, CIVS, and Election Runner show no reliable leakage. The Election Runner result warrants particular attention: while its accuracy  $t$ -test is significant ( $p < 0.001$ ), the AUC of 0.519 reveals that this is an artifact of prediction bias rather than genuine discrimination, a cautionary example of why accuracy alone is insufficient for evaluating side-channel leakage, and why AUC should be considered the primary metric.

The Helios and Belenios elections deliberately used extreme candidate name asymmetry to maximize any potential signal, yet neither produced above-chance classification. CIVS, with its simpler Condorcet-based ranking interface, similarly showed no signal. Election Runner, despite spanning 24 separate elections, produced only biased predictions without discriminative power. Crucially, all five platforms used the same asymmetric candidate naming scheme (CandidateA vs the 73-character Myxococcus name), ensuring that any name-length-dependent processing variation was equally available to the classifier across all platforms. The fact that only Polyas produced a genuine signal under these uniform conditions strengthens the conclusion that the leakage is platform-specific rather than an artifact of experimental design.

The architectural explanation is straightforward. In Helios and Belenios, the ballot is encrypted in the voter's browser before transmission. The ciphertext produced by ElGamal encryption is computationally indistinguishable regardless of the plaintext content, and the server processes all ciphertexts through the same code path without knowledge of the underlying vote. In Polyas, while client-side encryption is also employed, the server-side processing is opaque. The observed timing leakage suggests that some aspect of the server's handling of the encrypted ballot varies depending on its content, even if that variation is small.



# 6

## Discussion

This chapter interprets the experimental results presented in Chapter 5, examining the classifier’s behaviour across platforms, the practical significance of the detected leakage on Polyas, the limitations of the experimental approach, and potential countermeasures available to platform developers.

### 6.1 CLASSIFIER BEHAVIOUR ANALYSIS

An important distinction between positive and negative results lies in the behavior of the classifier during training. On Polyas, the train balanced accuracy was meaningfully above chance, indicating that the model could learn discriminative patterns from the training data and partially generalize to unseen elections. On Belenios, train balanced accuracy ranged from only 0.54 to 0.63 barely above the 0.50 baseline, indicating that the model could not learn from the data at all.

This distinction is critical in ruling out the alternative explanation that the classifier is simply too weak to detect a signal. The same LSTM architecture, with identical hyperparameters and training procedure, works on one platform and fails on the others. If the architecture were the limiting factor, it would fail uniformly. The platform-dependent success confirms that the classifier is responding to a genuine signal in the Polyas data that is absent in the Helios and Belenios data.

## 6.2 PRACTICAL IMPLICATIONS

The 57.1% accuracy achieved on Polyas corresponds to an improvement of 7.1 percentage points over random guessing. Although insufficient for reliably identifying an individual voter’s ballot choice, this level of leakage has implications for ballot secrecy that depend on the context.

**INDIVIDUAL-LEVEL INFERENCE.** An adversary targeting a specific voter can correctly infer their choice approximately 57% of the time. This is substantially better than chance, but far from deterministic. In a two-candidate election, the adversary’s confidence in any single inference remains low.

**POPULATION-LEVEL INFERENCE.** When an adversary has access to the traffic of many voters, even modest per-voter accuracy allows statistical inference about aggregate vote shares. An adversary observing 1,000 voters with 57% per-voter accuracy can estimate the true vote distribution with considerably more precision than random guessing would allow.

**BAYESIAN EXPLOITATION.** If an adversary has prior knowledge of the expected vote distribution (e.g., from public polls), even 57% accuracy allows meaningful Bayesian updating of beliefs about individual voters. In elections in which one candidate is expected to receive a large majority, the posterior probability of a correct inference for voters classified into the minority group can be substantially higher than 57%.

**COERCION AND VOTE BUYING.** In contexts where an adversary seeks to verify how a specific individual voted, for example, in a coercion or vote-buying scenario, 57% accuracy provides a weak but non-trivial verification signal. Over repeated elections, this signal could accumulate.

These considerations suggest that even modest timing leakage should be treated as a ballot secrecy concern, particularly in high-stakes elections or contexts where coercion is a realistic threat.

## 6.3 LIMITATIONS

Several limitations constrain the generalizability of these findings.

**NETWORK POSITION.** All experiments were conducted from a single network position (a home broadband connection via a VirtualBox VM). The timing characteristics observed may differ for adversaries positioned at other points in the network path, such as at an ISP, on a campus network, or on a corporate LAN. In particular, greater network distance would introduce additional jitter that could either obscure or amplify the timing signal.

**BINARY ELECTIONS.** All elections used exactly two candidates. Elections with more candidates may produce different traffic patterns and the classification task becomes harder as the number of classes increases. The binary setting represents the most favorable scenario for the attacker.

**SAMPLE SIZE.** The data set for each platform consisted of approximately 500 voters. While sufficient to detect statistically significant effects at the level observed for Polyas, a larger sample might reveal weaker effects on the platforms where no leakage was detected, or provide tighter confidence intervals for the positive result.

**SINGLE CLASSIFIER ARCHITECTURE.** The evaluation relies on a bidirectional LSTM. Although this architecture was validated by its ability to detect leakage on Polyas, a different model, for example, a one-dimensional CNN or a transformer might perform differently. The negative results on Helios and Belenios should be interpreted as “no leakage detectable by this classifier” rather than as a definitive proof of zero leakage.

**CONTROLLED ENVIRONMENT.** The automated voting setup introduces uniform browsing behavior among all voters, whereas real voters exhibit variable interaction speeds, hesitation, and revisiting of ballot options. Real-world traffic would contain more noise, which could reduce the effectiveness of the attack. In contrast, the controlled environment ensures that any detected signal originates from the platform itself rather than from user behavior, making the experimental setup appropriate for assessing platform-level vulnerabilities.

**TEMPORAL CONSISTENCY.** All voters within a single election were processed sequentially in a single session. Although candidate assignment was randomized to prevent temporal confounding, environmental factors such as server load variation, network congestion, or time-of-day effects could introduce noise that aids or hinders classification. The leave-one-election-out validation on Polyas mitigates this concern by testing generalization across sessions recorded at different times.

## 6.4 COUNTERMEASURE DISCUSSION

The results suggest several countermeasures that voting platform developers could adopt to mitigate timing-based information leakage:

**CONSTANT-TIME SERVER-SIDE PROCESSING.** The most direct countermeasure is to ensure that the server-side processing of encrypted ballots follows a constant-time code path regardless of the content. This is analogous to the constant-time implementations adopted in cryptographic libraries following the Brumley and Boneh timing attack [13]. Concretely, the server should perform identical operations for all ballot choices, avoiding any content-dependent branching, database operations, or validation logic.

**TRAFFIC PADDING.** Introducing random padding to TLS records or inserting dummy packets can obscure timing patterns. The effectiveness of padding against deep learning classifiers has been studied in the website fingerprinting literature: lightweight padding (WTF-PAD) was shown to be insufficient against the Deep Fingerprinting attack [18], while more aggressive schemes (Walkie-Talkie) were effective. Given the much weaker signal in the voting context compared to website fingerprinting, even lightweight padding may suffice.

**RANDOM SERVER-SIDE DELAYS.** Adding a random delay before sending the ballot confirmation would increase the noise in the timing channel. However, as Brumley and Boneh noted, random delays can be averaged out with sufficient measurements [13]. This countermeasure is therefore most effective when combined with rate limiting on the number of votes an observer can capture.

**CLIENT-SIDE ENCRYPTION WITH UNIFORM CIPHERTEXTS.** The negative results for Helios and Belenios suggest that the most robust architectural defense is to encrypt the ballot client-side using a cryptographic scheme that produces uniform ciphertexts. ElGamal homomorphic encryption achieves this naturally, as the ciphertext length and structure are independent of the plaintext. Platform designers should ensure that no server-side processing step reintroduces content-dependent timing after the encrypted ballot is received.

Belousova et al. [23] proposed and evaluated countermeasures against the related threat of participation and validity inference, quantifying the delay and memory overhead introduced by traffic padding. Their findings are complementary to ours: while their countermeasures

address coarser-grained inference (did the voter participate?), the same techniques, particularly traffic padding and constant-rate transmission, would also mitigate the finer-grained ballot content inference examined here.



# 7

## Conclusion

### 7.1 SUMMARY

This thesis investigated whether the ballot choices of voters in online voting systems can be inferred from timing patterns in encrypted network traffic. A reusable automated pipeline was developed that generates controlled elections with known ground truth, captures per-voter network traffic during vote submission, extracts packet-level timing sequences from the vote burst, and evaluates whether a bidirectional LSTM classifier can distinguish between ballot choices at a rate significantly above chance. The pipeline was applied uniformly to five online voting platforms: Polyas, Helios, Belenios, CIVS, and Election Runner.

The results demonstrate that timing-based ballot inference is feasible on at least one real-world platform. Polyas, a commercial closed-source system, exhibited statistically significant leakage with a classification accuracy of 57.1% ( $\text{AUC} = 0.595$ ,  $p = 0.004$ ). Detailed packet-level analysis localized the leakage to the server's 318-byte confirmation packet, where a 14.8 ms mean difference in inter-arrival time between candidates was observed across approximately 480 voters. The remaining four platforms Helios, Belenios, CIVS, and Election Runner did not show reliable leakage under identical experimental conditions, including deliberately asymmetric candidate names designed to maximize any potential signal.

The cross-platform comparison reveals a clear architectural pattern. The platforms that resisted the attack (Helios and Belenios) employ ElGamal homomorphic encryption with client-

side ballot encryption, producing uniform ciphertexts that the server processes identically regardless of content. Polyas, while also employing client-side encryption, evidently performs some content-dependent server-side processing that introduces measurable timing variation. The fact that the same classifier, features, and evaluation procedure produced a positive result on one platform and null results on four others confirms that the finding is platform-specific rather than a methodological artifact.

## 7.2 CONTRIBUTIONS

The primary contributions of this work are as follows.

1. A reusable methodology and automated pipeline for detecting timing-based side-channel leakage in online voting systems, applicable to any web-based voting platform with minimal adaptation.
2. Empirical evidence that the commercial voting platform Polyas exhibits a statistically significant timing side channel that leaks information about ballot content through encrypted network traffic.
3. Empirical evidence that four other platforms (Helios, Belenios, CIVS, and Election Runner) do not exhibit detectable leakage under the same experimental conditions, providing both negative results and a comparative baseline.
4. The identification of the specific network packet responsible for the leakage in Polyas (the 318-byte S2C confirmation), along with a statistical characterization of the timing difference (Cohen's  $d = 0.40$ , overlap coefficient = 0.56).
5. A methodological demonstration that accuracy alone is insufficient for evaluating side-channel leakage as shown by the Election Runner result, where a significant accuracy  $t$ -test masked a chance-level AUC caused by prediction bias and that AUC should be treated as the primary metric.

## 7.3 FUTURE WORK

Several directions could extend and strengthen these findings:

**ADVERSARIAL POSITION.** All experiments were conducted from a single home broadband connection. Testing from different network positions, such as a university campus network, a corporate LAN, or through a VPN, would establish whether the timing signal survives under different network conditions and latency profiles. An adversary positioned closer to the voting server (e.g., within the same data center or ISP) would experience less network jitter and may achieve higher classification accuracy.

**MULTI-CANDIDATE ELECTIONS.** This work evaluated only binary elections. Extending the analysis to elections with three or more candidates would test whether the leakage generalizes and how classification difficulty scales with the number of classes.

**REAL-WORLD TRAFFIC CONDITIONS.** The automated voting setup produced uniform browsing behavior. Evaluating the attack under realistic conditions, with variable user interaction speeds, concurrent network traffic, and various types of client device, would establish the practical exploitability of the leakage outside a controlled setting.

**COUNTERMEASURE EVALUATION.** The countermeasures discussed in the evaluation chapter (constant-time processing, traffic padding, random delays) could be implemented and empirically tested to quantify their effectiveness against the specific timing signal detected on Polyas.

**LONGITUDINAL ANALYSIS.** The experiments captured traffic at a single point in time. Platform updates, server infrastructure changes, or network configuration modifications could alter the timing characteristics. Periodic re-evaluation would establish whether the leakage is a persistent property of the platform architecture or a transient artifact of a specific deployment.

**INTEGRATION INTO DEVELOPMENT WORKFLOWS.** The pipeline is designed to be reusable and could be packaged as a testing tool for voting platform developers. Integrating it into a continuous integration workflow would allow timing leakage to be detected automatically during development and before deployment, rather than being discovered retrospectively.



# References

- [1] P. Ehin, M. Solvak, J. Willemson, and P. Vinkel, “Internet voting in estonia 2005–2019: Evidence from eleven elections,” *Elsevier*, 2022.
- [2] I.-P. U. Parline, “The estonian parliament,” 2023.
- [3] A. of Municipalities of Ontario, “Context for the 2022 ontario municipal election,” 2022.
- [4] T. Martínek and M. Malý, “Evaluation of compliance with democratic and technical standards of i-voting in elections to academic senates in czech higher education,” *arXiv*, 2025.
- [5] C. W. of America, “Resolution: Safe, secure, and private elections for national officers,” 2021.
- [6] Computershare, “Agm intelligence report 2023,” 2023.
- [7] U. N. G. Assembly, “Universal declaration of human rights,” 1948.
- [8] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine, “Privacy vulnerabilities in encrypted HTTP streams,” in *Proceedings of the 5th International Workshop on Privacy Enhancing Technologies (PET 2005)*. Springer, 2005, pp. 1–11.
- [9] M. Nasr, A. Bahramali, and A. Houmansadr, “DeepCorr: Strong flow correlation attacks on Tor using deep learning,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS ’18)*. ACM, 2018, pp. 1962–1976.
- [10] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, “Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations,” in *Proceedings of the 2008 IEEE Symposium on Security and Privacy (S&P ’08)*. IEEE, 2008, pp. 35–49.

- [11] J. K. Michael A. Specter and D. Weitzner, “The ballot is busted before the blockchain: A security analysis of voatz, the first internet voting application used in u.s. federal elections,” in *Proceedings of the 29th USENIX Security Symposium*, U. Association, Ed., 2020, available at <https://www.usenix.org/system/files/sec20-specter.pdf>.
- [12] P. C. Kocher, “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems,” in *Advances in Cryptology — CRYPTO ’96*, ser. LNCS, vol. 1109. Springer, 1996, pp. 104–113.
- [13] D. Brumley and D. Boneh, “Remote timing attacks are practical,” in *Proceedings of the 12th USENIX Security Symposium*. USENIX Association, 2003.
- [14] B. B. Brumley and N. Tuveri, “Remote timing attacks are still practical,” in *Computer Security — ESORICS 2011*, ser. LNCS, vol. 6879. Springer, 2011, pp. 355–371.
- [15] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, “Website fingerprinting in onion routing based anonymization networks,” in *Proceedings of the 10th ACM Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2011, pp. 103–114.
- [16] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, “Effective attacks and provable defenses for website fingerprinting,” in *Proceedings of the 23rd USENIX Security Symposium*. USENIX Association, 2014, pp. 143–157.
- [17] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, “Automated website fingerprinting through deep learning,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2018.
- [18] P. Sirinam, M. Imani, M. Juarez, and M. Wright, “Deep fingerprinting: Undermining website fingerprinting defenses with deep learning,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS ’18)*. ACM, 2018, pp. 1928–1943.
- [19] S. Bhat, D. Lu, A. Kwon, and S. Devadas, “Var-CNN: A data-efficient website fingerprinting attack based on deep learning,” *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 4, pp. 292–310, 2019.
- [20] M. S. Rahman, P. Sirinam, N. Mathews, K. G. Gangadhara, and M. Wright, “Tik-Tok: The utility of packet timing in website fingerprinting attacks,” in *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 3, 2020, pp. 5–24.

- [21] R. Schuster, V. Shmatikov, and E. Tromer, “Beauty and the burst: Remote identification of encrypted video streams,” in *Proceedings of the 26th USENIX Security Symposium*. USENIX Association, 2017, pp. 1357–1374.
- [22] A. Reed and M. Klimkowski, “Leaky streams: Identifying variable bitrate DASH videos streamed over encrypted 802.11n connections,” in *Proceedings of the 13th IEEE Consumer Communications & Networking Conference (CCNC)*. IEEE, 2016.
- [23] A. Belousova *et al.*, “Inference attacks on encrypted online voting via traffic analysis,” *arXiv preprint arXiv:2509.15694*, 2025, available at <https://arxiv.org/abs/2509.15694>.
- [24] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, “Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges,” *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.
- [25] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, “Deep packet: A novel approach for encrypted traffic classification using deep learning,” *Soft Computing*, vol. 24, pp. 1999–2012, 2020.
- [26] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, “Network traffic classifier with convolutional and recurrent neural networks for Internet of Things,” *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017.



# Acknowledgments

Thank you to Anastasiia and Francesco for all their invaluable help throughout this project