The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in U.S. Federal Elections*

Michael A. Specter
MIT†

James Koppel
MIT‡

Daniel Weitzner
MIT§

Abstract

In the 2018 midterm elections, West Virginia became the first state in the U.S. to allow select voters to cast their ballot on a mobile phone via a proprietary app called "Voatz." Although there is no public formal description of Voatz's security model, the company claims that election security and integrity are maintained through the use of a permissioned blockchain, biometrics, a mixnet, and hardware-backed key storage modules on the user's device. In this work, we present the first public security analysis of Voatz, based on a reverse engineering of their Android application and the minimal available documentation of the system. We performed a clean-room reimplementation of Voatz's server and present an analysis of the election process as visible from the app itself.

We find that Voatz has vulnerabilities that allow different kinds of adversaries to alter, stop, or expose a user's vote, including a sidechannel attack in which a completely passive network adversary can potentially recover a user's secret ballot. We additionally find that Voatz has a number of privacy issues stemming from their use of third party services for crucial app functionality. Our findings serve as a concrete illustration of the common wisdom against Internet voting, and of the importance of transparency to the legitimacy of elections.

1 Introduction

In 2018, Voatz, a private Boston-based company, made history by fielding the first Internet voting app used in high-stakes U.S. federal elections. Mainly targeting overseas military and other absentee voters, Voatz has been used in federal, state, and municipal elections in West Virginia, Denver, Oregon, and Utah, as well as the 2016 Massachusetts Democratic Convention and the 2016 Utah Republican Convention [38]. The company has recently closed a $7-million series A [22], and is on track to be used in the 2020 Primaries.

In this paper, we present the first public security review of Voatz. We find that Voatz is vulnerable to a number of attacks that could violate election integrity (summary in Table 1). For example, we find that an attacker with root access to a voter's device can easily evade the system's defenses (§5.1.1), learn the user's choices (even after the event is over), and alter the user's vote (§5.1). We further find that their network protocol can leak details of the user's vote (§5.3), and, surprisingly, that that the system's use of the blockchain is unlikely to protect against server-side attacks (§5.2). We provide an analysis of these faults, and find that exploitation would be well within the capacity of a nation-state actor.

While the introduction of Internet voting in the U.S. is relatively new, the history surrounding electronic-only voting is not. In the wake of counting errors, recount discrepancies, and uninterpretable ballots wreaking havoc during the 2000 U.S. Presidential race, Congress passed the Help America Vote Act (HAVA) [49], a bill targeted toward helping states move away from outdated and problematic punchard-based systems. The Election Assistance Commission (EAC), a new executive agency created by HAVA, was charged with distributing these funds, and has since provided over $3.3 billion to various states to help improve election infrastructure [25].

Unfortunately, HAVA lacked stringent guidelines on what replacement systems were allowed to be purchased. As a result, many states acquired unvetted electronic-only voting machines, known as Direct-Recording Electronic (DRE) systems. Numerous studies have since shown DRE systems that lack a paper backup are extremely vulnerable to a wide range of attacks, allowing adversaries to surreptitiously change the outcome of an election [17, 27, 64].

Today, we are witnessing similar developments in response to Russia's interference in the 2016 election. Bills have been introduced in both the Senate [41] and House [59] that aim to provide funding to revamp election infrastructure. At the same time, there has been renewed interest in cryptography due to recent advances in accountable and transparent systems such

*With appreciation to Barbara Simon [39]

†EECS PhD Candidate, CSAIL, Internet Policy Research Initiative

‡EECS PhD Candidate, CSAIL, Computer Assisted Programming Group

§Research Scientist, CSAIL, Internet Policy Research Initiative

Adversary	Attacker Capability				
	Suppress Ballot	Learn Secret Vote	Alter Ballot	Learn User's Identity	Learn User IP
Passive Network (§5.3)		✓			✓
Active Network (§5.3)	✓	✓			✓
3rd-Party ID Svc. (§5.4)	✓			✓	✓
Root On-Device (§5.1)	✓	✓	✓	✓	✓
Voatz API Server (§5.2)	✓	✓	✓	✓	✓

Table 1: Summary of Potential Attacks by Adversary Type: Here we show what kind of adversary is capable of executing what sort of attack; e.g. a Passive Network adversary is capable of learning a user's secret ballot, and the User's IP. The viability of some of these attacks are dependent on the configuration of the particular election, (the ballot style, metadata, etc.), see the relevant section listed for explicit details.

as the blockchain [48], and the proliferation of mobile devices carrying hardware-backed secure enclaves for cryptographic operations as well as biometrics.

The result is increased speculation about how mobile devices can be used to safely allow for voting over the Internet. At the time of writing there are at least four companies attempting to offer internet or mobile voting solutions for high-stakes elections [47], and one 2020 Democratic presidential candidate has included voting from a mobile device via the blockchain in his policy plank [8]. To our knowledge, only Voatz has successfully fielded such a system.

Unfortunately, the public information about Voatz's system is incomplete. Voatz's FAQ [5], blog, and white paper [42] provide only a vague description of their overall system and threat model; Voatz claims it leverages some combination of a permissioned blockchain, biometrics, and hardware-backed keystores to provide end-to-end encrypted and voter verifiable ballots. However, despite calls to release a more detailed analysis and concerns raised by many in the election security community [24, 50], as well as elected representatives [54], Voatz has declined to provide formal details, citing the need to protect their intellectual property [60]. Worse, when a University of Michigan researcher conducted dynamic analysis of the Voatz app in 2018, the company treated the researcher as a malicious actor and reported the incident to authorities. This resulted in the FBI conducting an investigation against the researcher [36, 40, 43, 63].

This opaque stance is a threat to the integrity of the electoral process. Given the contentious nature of high-stakes elections,[1] the stringent security requirements of voting systems, and the possibility of future interference by foreign government intelligence agencies, it is crucial that the details of any fielded election system be analyzable by the public. The legitimacy of the government relies on scrutiny and transparency of the democratic process to ensure that no party or outside actor can unduly alter the outcome.

Methodologically, our analysis was significantly complicated by Voatz's lack of transparency — to our knowledge, in previous security reviews of deployed Internet voting systems (see Switzerland [34], Moscow [30], Estonia [57], and Washington D.C. [62]), researchers enjoyed significant information about the voting infrastructure, often including the system's design and source code of the system itself.

We were instead forced to adopt a purely black-box approach, and perform our analysis on a clean-room re-implementation of the server gained by reverse engineering Voatz's publicly available Android application. We show that, despite the increased effort and risks to validity, our analysis is sufficient to gain a fair understanding of Voatz's shortcomings. In particular, we demonstrate that our attacks stand up against optimistic assumptions for the unknown parts of Voatz's infrastructure (see §5). We hope that this work serves as a useful discussion point for policy makers and future researchers, and can be used to encourage system developers to be more transparent.

The rest of the paper is organized as follows: We begin in §2 with short background on the security requirements of elections, Voatz's claims of security, and known work analyzing Voatz. We continue in §3 by describing our reverse engineering methodology, and discus how we minimize threats to validity. In §4, we illustrate Voatz's system as discovered in our methodology, including all parts of the voting process, the server infrastructure, custom cryptography used, and provide a brief discussion of factors we were unable to confirm in our analysis. Next, §5 enumerates the attacks discovered in our analysis of Voatz. We conclude with a discussion in §6 to provide lessons learned and recommendations for policymakers in this space moving forward.

2 Background

In this section we describe some of the security requirements commonly seen in proposed cryptographic voting systems.

[1] We refer to high-stakes elections as those where adversaries are likely willing to expend resources to alter the course of an election. Certain elections, like student governments, clubs, and online groups are generally considered "low stakes," where federal or municipal elections are "high-stakes." This is consistent with the research literature on the subject (see, e.g. [7]).

We then discuss the claims made by Voatz, and conclude by providing an overview of prior analyses of Voatz.

Voting as a research subject in both applied vulnerability discovery and in cryptography is not new. Below is a short description of security definitions commonly used in the voting system literature.

Correctness and usability: To ensure the legitimacy of the election, a voting system must convincingly show that all eligible votes were cast as intended, collected as cast, and counted as collected [15].

Secret Ballot: A secret ballot requires that 1) No voter is able to prove her selections, and 2) no voter's choices can be surreptitiously released or inferred. The goal of a secret ballot is to provide an election free from undue influence: if a voter is able to prove the way they voted, they can sell their vote, and if a voter's preferences are leaked they may suffer harassment and coercion [16].

End-to-End Verifiability: End-to-End (e2e) verifiable voting systems have the property that voters receive proof that their selections have been included, unmodified, in the final tallying of all collected ballots, *without the need to trust any separate authority to do so*. There have been research prototypes developed that provide such guarantees while maintaining a secret ballot, using techniques such as visual cryptography, homomorphic cryptography, invisible ink, and mixnets [13, 19–21, 55].

2.1 Voatz's Claims of Security

Although there is no public, formal description of their system, Voatz does make a number of claims about their system's security properties via their FAQ [5]. We provide further quotes and analysis on these claims in Appendix A, and a short summary below:

Immutability via a permissioned blockchain: Voatz claims that once a vote has been submitted, Voatz uses "...blockchain technology to ensure that...votes are verified and immutably stored on multiple, geographically diverse verifying servers." The FAQ goes into further detail, discussing the provision of tokens for each ballot measure and candidate.[2]

End-to-End vote encryption: Voatz makes multiple references to votes themselves being encrypted "end to end." [3] To the author's knowledge, there is no formal definition of "end to end vote encryption;" for example, it is unclear where the "ends" of an end to end encrypted voting scheme are. It is worth noting that there exist homomorphic cryptography schemes that tally votes over the vote ciphertexts, so that one need only decrypt the an aggregate vote, maintaining individual voter privacy [14], but it is unclear from the FAQ if this is what Voatz intends.

Voter anonymity: Voatz claims that "the identity of the voter is doubly anonymized" by the smartphone and the blockchain, and that, "Once submitted, all information is anonymized, routed via a 'mixnet' and posted to the blockchain." [4]

Device compromise detection: Voatz claims to use multiple methods to detect if a device has been jailbroken or contains malware, and that "The Voatz app does not permit a voter to vote if the operating system has been compromised."[5]

Voter Verified Audit Trail: Voatz claims that voters receive a cryptographically-signed digital receipt of their ballot after their vote has been submitted.[6] The guarantees of such a receipt are unclear, although, perhaps this is meant to provide similar guarantees as seen in the End-to-End verifiable cryptosystems mentioned above.

2.2 Prior Scrutiny of Voatz

While we are the first to publish an in-depth analysis of Voatz, others have raised concerns about their system, security claims, and lack of transparency. Jefferson et al [37] compiled a long list of unanswered questions about Voatz, including the app's use of a third party, Jumio, as an ID verification service. Several writers observed the election processing and audit of the Voatz pilot during the 2019 Denver Municipal elections, and found that the main activity of the audit was to compare a server-generated PDF of a voter's ballot with the blockchain block recording the same [35, 58]. Kevin Beaumont found what appeared to be several Voatz service-related credentials on a public Github account [11], and that the Voatz webserver was running several unpatched services [12]. Voatz responded citing a report from the Qualys SSL checker as evidence of the site's security [46], and later claimed that the insecure server Beaumont identified was an intentionally-insecure "honeypot operation" [61]. As a result of this public scrutiny, in November 2019, U.S. Senator Ron Wyden called on the NSA and DoD to perform an audit of Voatz [54].

3 Experimental Methodology

As performing a security analysis against a running election server would raise a number of unacceptable legal and ethical

[2] Sections "How do we know if the Voatz app can be trusted?" and "How does the Voatz blockchain work?"

[3] Sections "If a user's phone or mobile network is compromised, is their vote compromised as well?" and "What happens if the smartphone is hacked?"

[4] Sections "How is anonymity preserved?" and "How do I vote?"

[5] Sections "If a user's phone or mobile network is compromised, is their vote compromised as well," "What happens if the smartphone is hacked," and "If a user's phone or mobile network is compromised, is their vote compromised as well?"

[6] Sections "How do we know if the Voatz app can be trusted," "Is there a paper trail," and "How can votes stored on the blockchain be audited?"

concerns [53], we instead chose to perform all of our analyses in a "cleanroom" environment, connecting only to our own servers.[7] Special care was taken to ensure that our static and dynamic analysis techniques could never directly interfere with Voatz or any related services, and we went through great effort so that nothing was intentionally transmitted to Voatz's servers.[8]

To gain a better understanding of Voatz's infrastructure, we began by decompiling the most recent version of their Android[9] application as found on the Google Play Store as of January 1, 2020[10] and iteratively re-implemented a minimal server that performs election processes as visible from the app itself. This included interactions involved in device registration, voter identification, and vote casting. We used two devices for our dynamic analysis and development: a Voatz-supported Pixel 2 XL running Android 9, and a Voatz-unsupported Xiaomi Mi 4i running the Lineage OS with Android 8, both jailbroken with the Magisk framework [2].

In order to redirect control to our own server, we were forced to make some small changes to the application's control flow. To reduce threats to validity, we limited these modifications to the minimum necessary in order to redirect all network communication. We:

1. Disabled certificate pinning and replaced all external connections to our own servers;
2. Disabled the application's built-in malware and jailbreak detection. Details are available in §5.1.1; and,
3. Removed additional encryption between the device and all still active third parties, re-targeted all communication from these services with our own server, and reimplemented the necessary parts of their protocols as well.

While all of this could have been accomplished by statically modifying the program's code, we instead opted to dynamically modify or "hook" relevant parts of the code at runtime. Modifications therefore required no changes to the application code itself, only to code running on our test devices, allowing for rapid development and transparency about what was modified at each stage of our analysis.

Despite this lengthy description, our codebase is relatively simple. The on-device hooking code consists of ~500 lines of Java that leverages the Xposed Framework, a series of hooking libraries that are well supported and popular in the Android modding community. Our server implementation is ~1200 lines of code written in Python using the Flask web framework. While we do intend to release our source, we are withholding code to provide time until Voatz and others can respond to the concerns raised in this paper.

[7] Unless otherwise specified, throughout this paper, any reference to communication we performed with "a server" or "the server" refers to our own server infrastructure.

[8] Indeed, at the time of analysis, Voatz's servers appeared to be down including for an unmodified app running on an supported and up-to-date device.

[9] We did no analysis on and make no claims about Voatz's iOS app.

[10] Version 1.1.60, SHA256 191927a013f6aae094c86392db4ecca825866ae62c6178589c02932563d142c1

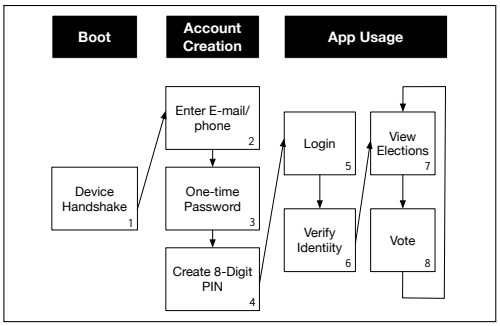


Figure 1: Voatz's workflow as seen from the device.

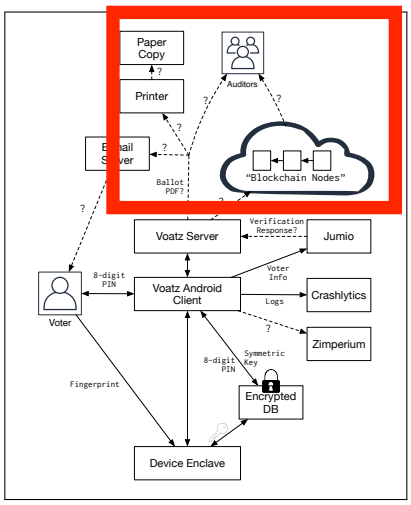


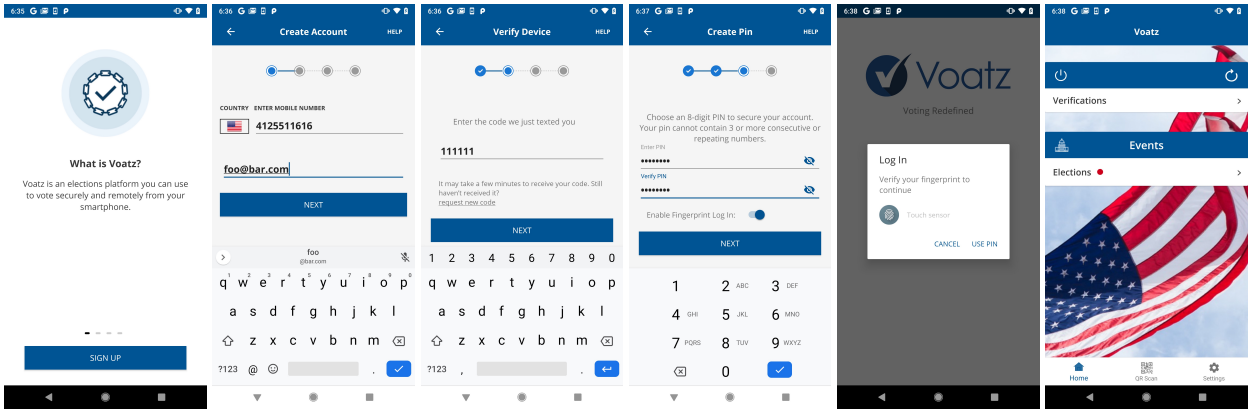
Figure 2: Dataflow between Voatz components and external services. Dashed lines are believed to exist but have not been directly observed.

4 Voatz's System Design

In this section, we present Voatz's infrastructure as recovered through the methodology presented in §3. We begin with an overview of the system §4.1, illustrating the process by which a user's device interacts with the app during all stages of the voting process including Voatz's custom cryptographic protocol §4.1.1, user registration and voter verification §4.2, and vote casting §4.3. Finally, we discuss all non-protocol device-side defensive measures we discovered §4.4.

4.1 Process Overview

Figure 1 presents a diagram of the steps that occur in-app from login to election voting. They are:



(a) Initial screen. (b) Email and Phone. (c) OTP Verification. (d) Pin number. (e) First login. (f) Main screen.

Figure 3: The user registration process, connecting to our server reimplementation.

1. The device initiates a handshake with the server, creating a shared key which enables an extra layer of encryption beyond TLS (Box 1). Communication between the device and Voatz server is described in more detail in §4.1.1.
2. The user creates an account by providing their E-mail address, phone number, and an 8-digit PIN (Boxes 2-4).
3. The user logs in with this PIN (Box 5).
4. The user verifies their identity, using Voatz's integration with a third-party service called Jumio (Box 6). The app requests a scan of the user's photo ID, a recording of their face, and the user's address, and then sends all of this information to Jumio's servers for verification and OCR.
5. The user selects from a list of open elections, and then marks and submits their ballot. Depending on the election configuration, Voatz can allow "vote-spoiling,"[11] so this process may be repeated prior to the election closing. (Boxes 7-8)

Communication Figure 2 shows the communication between components of Voatz and other entities. As we were only able to directly observe communication involving the Voatz app, the rest of this diagram is an attempted reconstruction based on documents released by Voatz [42] and by the Denver Elections Division [29].

The three primary third-party services used by the Voatz app are the identify-verification service Jumio, a crash reporting service Crashlytics, and a device security service Zimperium. Of these, the most significant is Jumio, which Voatz relies on for ID verification, and to which the app sends substantial personal information (see §4.2).

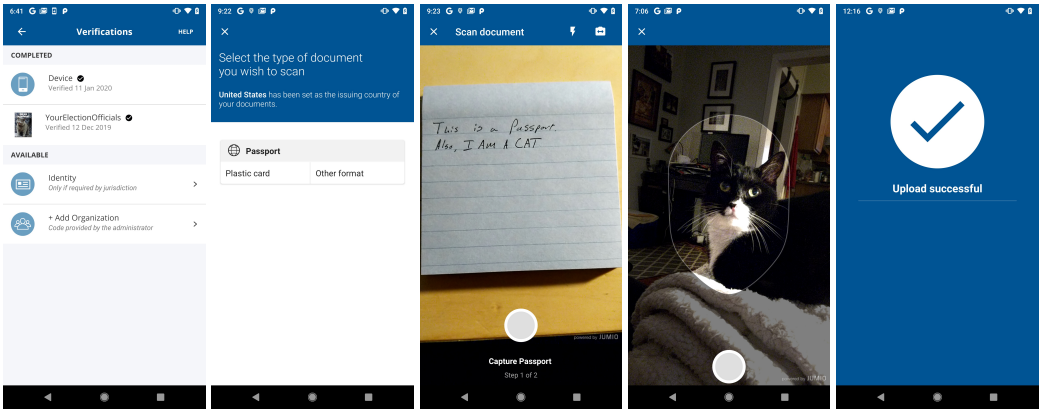
[11] Vote spoiling refers to casting a new vote that invalidates all previously cast ballots.

4.1.1 Voatz Server Handshake and Protocol

Voatz's server is implemented as a REST application — all communication between Voatz's server and the application occur as a series of JSON-encoded HTTPS GET, PUT, and POST commands. The app's REST server is `voatzapi.nimsim.com`, with `voatz.com` only used for static assets such as images and text. All parts of the protocol leverage the Android OS's built-in TLS stack, and uses certificate pinning to ensure that the incoming certificate is from a particular issuing Certificate Authority.

Next, *on top of TLS*, the system performs a "device handshake" with the following steps:

1. The App generates 100 ECDSA SECP256R1 keypairs, and sends the Server all 100 corresponding public keys. The device saves only the 57th keypair (PK_D, SK_D).
2. The Server generates 100 ECDSA SECP256R1 keyparis, selects the 57th (PK_S, SK_S), and performs the rest of an ECDH key exchange to generate a shared secret (SK_{ecdh}).
3. The Server generates AES-GCM parameters; a random AES-GCM 256-bit symmetric key (SK_{aes}), a random 16-bit nonce (N), and a Tag (T).
4. The Server then sends the device:
   - The 100 public keys generated above, including the PK_S as the 57th key.
   - ECDH-Encrypt$(SK_{ecdh}, SK_{aes}||N||T)$
5. Out of the 100 public keys sent by the Server, the App selects the 57th pubkey (PK_S), and finishes the ECDH handshake to create the ECDH shared key SK_{ecdh}. Finally, it decrypts and parses the AES-GCM parameters(SK_{aes}, N, T).



(a) Verification fragment. (b) Document select (c) Picture of an ID. (d) Face "selfie." (e) Verification success.

Figure 4: The voter verification process as seen from our experimental environment.

This handshake is performed every time the app is launched for the first time, and, from this point forward in the app's execution, *every communication* between the App and the Server is encrypted using the standard AES-GCM algorithm by way of SK_{aes}, in addition to the encryption provided by TLS. Note that there is no authentication of the ECDSA keys by the app, beyond the encapsulating TLS certificates. This made it very simple to retarget the server — we replaced all required URLs in-app to our own and followed the protocol. Further, this renders the use of the handshake somewhat unclear, as it offers no protection against active MITM attacks over the authentication already provided by TLS.

It also is worth mentioning that all but the 57th keys are abandoned immediately on the device side — both the extraneous secret keys the device generated in the first step and the public keys it receives from the server. We conclude that this 100-key exchange is likely a attempt at obfuscation, rather than serving any useful purpose to the security protocol.

4.2 User Registration & ID Verification

After the app has completed the device handshake, the user can begin the registration process, which can be seen in Figure 3. Here the user is asked to submit their email and phone number, and perform a One Time Password operation via SMS. Finally, the user selects an 8-digit PIN number which is then sent to the server, and used extensively in user authentication.

If the user has a fingerprint registered with their device, they are given the option to "enroll" their fingerprint as an alternative authentication mechanism. Effectively, this works by storing the PIN on-disk, encrypted using a key biometrically tied to the user's fingerprint via the Android Keystore API.

The Android Keystore is a system service that, if used correctly,[12] will perform various cryptographic operations on behalf of the application, on application-level data, *without* exposing the requisite key material to the application's host memory.[13] Further, when supported by the device's hardware, these device-level keys are stored in the manufacturer's protected hardware, and can be made to require the user to enter in their device password or fingerprint before they are used.

After registration, the user is asked to log in via the PIN (or fingerprint decryption of the PIN). In addition to the PIN, there are four pieces of information sent to the server to authenticate the user at log in: a unique device ID generated via Android's ANDROID_ID system,[14] a customer ID number, a "nextKey" value, and an "auditToken". The nextKey and auditToken are originally received from the API server, are never modified except when updated by the server, and do not appear to be used in any device-side cryptography. How these authentication parameters are stored is explored in §4.4.

After authentication, the user may still need to provide some proof of identity, which requires visiting the verification menu from the main screen (Figure 4a). When the user selects the identity option, the app launches Jumio's sub-activity to select a document type (Figure 4b[15]). The user is prompted to take a photo of their ID or Passport (4c), and to take a selfie photo (4d), after which a dialog prompts the user for their registered voting address (not pictured). The app then uploads data to Jumio's server, including the user's photo, the voter's name, address, and photo ID (4e).[16] Finally, after receiving a

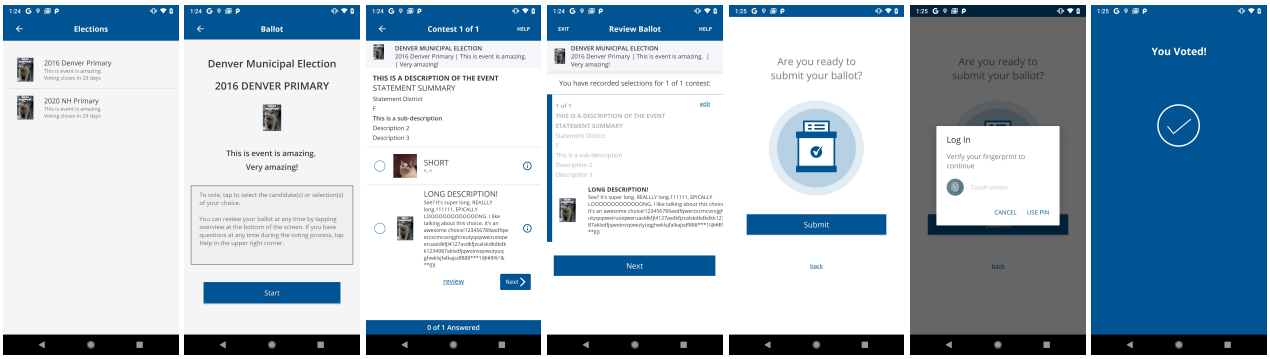
[12] In §5.1.2, we will see that the app does not use the Keystore correctly.

[13] See Android's Keystore documentation for details [9].

[14] See [10] for more information about Android's local device UUIDs.

[15] These options are based on metadata about available documents sent from our mock implementation of Jumio's servers. This screen may differ when the app is connected to the genuine Jumio servers.

[16] Furthermore, Jumio itself has disclosed that it uses a third party, Facetec, to help analyze the video selfies [6]. As we do not have visibility into their services, we cannot confirm whether or not Jumio actually transmits information to Facetec-controlled servers.



(a) Event selection. (b) Ballot. (c) Question. (d) Review. (e) Submission. (f) PIN Decryption. (g) Success.

Figure 5: The voting process as seen in a mock election we created for this experiment.

response from Jumio's server, the app sends a subset of the user's data to Voatz's server as well.

It is worth noting that the small, translucent logo in the bottom right corner of the photos taken during this process (Figures 4c, 4d) appears to be the only in-app indication to the user that Jumio exists. This is the only way by which a user might be aware that sensitive data is being sent to a 3rd party.

4.3 Vote Casting

After the user is verified, the app queries the server for configuration data relating to what events the voter is allowed to participate in, activating a menu for the user to select from available events (see Figure 5). This configuration data includes all events to which the voter has access, those event's ballots, each ballot's particular questions, and the options available for those questions.

The voter begins by selecting an event (5a), and is then able to view questions associated with these particular events, select responses (or no response at all, depending on the event configuration), and submit their response to the server. At the point of submission, the user is again asked to decrypt their PIN (5e), which is used as a final authentication mechanism before the ballot is submitted to the server.

It is important to note that the vote is *not* submitted directly to any blockchain-like system, and is instead submitted via this API server. Additionally, although the user is asked to authenticate before submission, beyond the MAC associated with the AES-GCM algorithm and enclosing TLS session, the text of the vote itself is not otherwise signed. The only indication of blockchain-like tokens being submitted or exchanged is the "auditToken", but this string is never altered by the app, and appears to be a single, static value. Figure 6 shows the entirety of what is sent to the server, AES encrypted, after a user submits their vote.


```
{ "voteData": [
    {
      "summary": "Best cat?",
      "questionId": "1",
      "isRCVFlag": false,
      "isRCV": false,
      "description1": "bogus desc",
      "statements": [
        {
          "summary": "Statement Summary",
          "statementId": "Statement ID",
          "description3": "Description 3",
          "choices": [
            {
              "choiceDetails": {
                "imageUrl": "https://bit.ly/36DJbC4",
                "webUrl": "https://bit.ly/36DJbC4"
              },
              "choiceId": "1",
              "description1": "^",
              "nonSelectable": false,
              "description2": "^",
              "description": "Short"
            }
          ],
          "description1": "This is a sub-description",
          "description": "This is a description of the event",
          "maxSelect": "1",
          "gender": "F",
          "description2": "Description 2",
          "district": "Statement District"
        }
      ],
      "description3": "bogus desc",
      "description2": "bogus desc",
      "description": "bogus desc"
    }
  ],
  "auditToken": "SomeAuditTokenValue",
  "controlNumber": "1",
  "customerId": 267732387,
  "eventId": 1 }
```


Figure 6: The above is the entirety of the decrypted payload for a vote submission in our synthetic election.

4.4 Device-Side Defensive Measures

In the process of performing our analysis we discovered that Voatz employs a number of obfuscation techniques, leverages a third party virus scanning service, and uses an encrypted database to protect locally stored sensitive data. We explore each below:

On-disk encrypted database: After the registration has been completed, the user's login credentials (the nextKey, auditToken, and customer ID number), as well as the voter's entire vote history, are stored in an encrypted database using the Realm database framework [3]. When Voatz's app attempts to query the database, the Keystore asks the user to authenticate via a fingerprint or PIN (see Figure 5f), before performing the required operations.

The key for the database is linked directly to the user's PIN; specifically, the system runs PBKDF2 with SHA1 over the PIN to generate the key. Recall that this allows the system to use a fingerprint as an alternative method of decrypting the database — At log in, the app can authenticate via the fingerprint to decrypt the PIN, or use the PIN directly to decrypt the database and gain access to the rest of the app.

Third-party Malware Detection (Zimperium): Voatz leverages a third-party antivirus solution called Zimperium. At initialization time, the Voatz app loads Zimperium's code as a separate service and registers a series of callbacks that will alert the API Server if Zimperium detects a threat. This message includes the details of the threat, the user ID, and device ID, and the IP address of the offending device.

Zimperium's scans include (but are not limited to) known exploit proofs of concept, known malware, and indicators that the user has installed known superuser tools indicative of a rooted / jailbroken device. Additionally, Zimperium will trigger callbacks if the user appears to have enabled Android's local debugging features such as remote `adb` debugging.

Partial Code Obfuscation and Packing: Without the developer taking extra precautions, Android apps may be readily unpacked and decompiled to near the original source via easy to use tools such as APKTool [1] and JADX [56]. However, much of the Voatz app is obfuscated using a packer that presents several barriers to analysis.

First, many of the classes and function names were renamed to random Unicode strings. Beyond making the resulting decompilation more difficult to read, this obfuscation also caused APKTool to crash, while JADX successfully completed decompilation, but left many of the resource files (including application strings and images) unreadable. Voatz's app also contained a few zip files that appear to perform a zip bomb attack [28], which defeats some implementations of `unzip`. Finally, all included 3rd-party native libraries for ARM failed to open in our version of IDA, although it is unclear if this was an active defensive measure as they were successfully disassembled using Ghidra.

We were able to defeat the obfuscation by intensive manual analysis and, in some cases, were aided in recovering the original variable names by the app itself. First, the app uses many libraries which internally depend on Java reflection, rendering the obfuscator unable to rename any classes or methods referenced in this way. Second, the app and some of its libraries are written in Kotlin. While some Kotlin idioms do not decompile easily to Java, the use of Kotlin overall aids reverse-engineering — the Kotlin compiler inserts many runtime checks into the code, each including a string with an error message to display in case of failure. The class, function, and variable names are often stored in these strings.

String Obfuscation To further complicate static analysis, the strings that control cryptographic parameters of the device handshake (e.g. "AES-GCM") are obfuscated with an XOR-based scheme and then automatically deobfuscated at runtime. As the strings hidden in this way include error messages generated by the Kotlin compiler, this appears to be the result of an automated tool that had been enabled for only these particular methods.

4.5 Unconfirmed Portions of the Process

As we lack access to Voatz's servers and deliberately avoided any interaction with them, there are unfortunately a few instances of where we are unable to confirm how certain third-party actors in the system behave.

Zimperium execution confirmation: Zimperium may communicate back to its own servers confirming that the service is running, and then communicate if Zimperium is active directly to Voatz. To the best of our knowledge, there is no public documentation that suggests this is how Zimperium works, and we find no indication from the callbacks associated with Zimperium that this is occurring. Further, any such mechanism could be similarly subverted with the techniques applied in §5.1.1, though likely with more effort.

Jumio voter confirmation: Jumio's documentation discusses at length the ability to have Jumio's servers communicate for out-of-band verification of a user. Since this is well documented, we therefore assume that Voatz's API server is communicating directly with Jumio for ID verification.

Ballot Receipts and the Blockchain: According to a Voatz whitepaper, votes are recorded on a 32-node permissioned blockchain spread across multiple Amazon AWS and Microsoft Azure datacenters [29]. Footage of the audit of the 2019 Denver Municipal elections shows that the auditing

process consists of manually decrypting blockchain blocks indicating transactions, obtaining several fields including a hash of the voter's choices. The auditor then manually compares the hash via a lookup table to a PDF displaying the voter's choices. These PDFs are allegedly also printed out by the election authority as a paper record, and are redacted versions of the receipt E-mailed to voters. While we know that, in the Denver election, many voters manually replied to indicate that they received a receipt, there is no evidence that Voatz can automatically verify receipt delivery [35].

In our exploration of the code, we find no indication that the app receives or validates any record that has been authenticated to, or stored in, any form of a blockchain. We further found no reference to hash chains, transparency logs, or other cryptographic proofs of inclusion. We conclude that any use of a blockchain by Voatz likely takes place purely on the backend, or in the receipt stage via the use of some other mechanism.

The only references to voter receipts in-app come from a dialog that requests a passcode from the server, and an (apparently unimplemented) QR code reader. The text of the voter receipt dialog appears to confirm that ballot receipts are indeed sent to the voter via email, and encrypted with a passcode (see Figure 7). Voatz's QR code reader has functional code for an out-of-band method of receiving organization IDs, which allows the voter to participate in particular events, and a largely unimplemented stub for verifying a vote — attempting to scan a QR code that would start the process of vote verification will result in the "not yet supported" message presented in Figure 7.

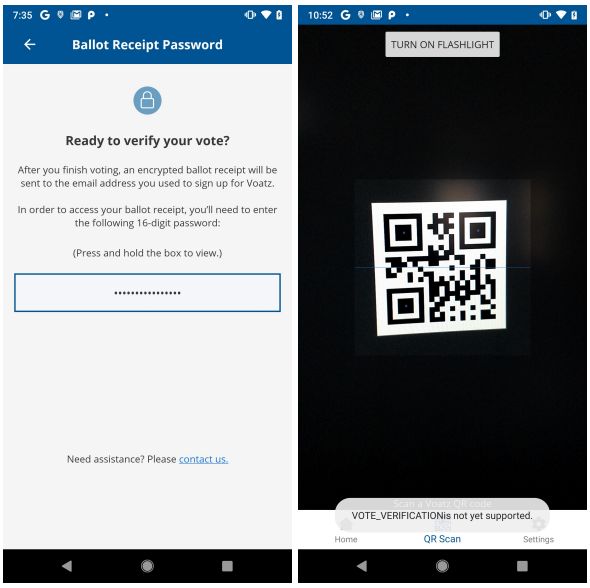


Figure 7: Left: the password request screen. Right: the QR code capture screen; note the popup indicating that the VOTE_VERIFICATION QR code type is unimplemented.

5 Analysis and Attacks

In this section, we explore various attacks assuming the role of an adversary that has control over particular parts of the election system. This includes three adversaries with various levels of access to individual parts of the overall infrastructure:

1. An attacker that has gained control of a user's device,
2. An attacker that has gained control over Voatz's API server, and
3. A network adversary that can intercept all network activity between voter's device and the API server, but has no key material or other access to the voter's and Voatz's systems.

We believe these adversaries to be credible given the high-stakes nature of the elections in which Voatz is intended to be used, and the resources of the associated attackers. Gaining root control of a user's device can happen through any number of means requiring various levels of skill — via malware, an intimate partner or spouse, as part of a border crossing, etc. Network adversaries could come in similarly many forms, including those that exploit a user's home router (which are notoriously insecure [32, 33]), the unencrypted coffee shop wifi a user attempts to vote from, or the user's ISP.

Including Voatz's API server in this analysis is useful for a number of reasons. While accessing Voatz's server may be more difficult than the user's device and/or the network infrastructure between the server and the user, if the use of Voatz were to be raised to the point that their userbase may alter the outcome of an election, it is not impossible for them to be the target of nation-states, at which point, it is also not outside of the realm of possibility that intelligence agencies would expend considerable resources, leveraging undisclosed 0-day vulnerabilities, espionage, coercion, or physical attacks, to gain access to crucial systems or key material. Further, a key promise of the blockchain is that it provides an environment where the voter and election authority may be able to trust the system, rather than just Voatz, that the election was conducted correctly.

Assumptions & Threats to Validity As we lack concrete implementation details about the server infrastructure or backend, we cannot make assumptions about what Voatz logs to their blockchain, the operational security of their servers, blockchain, or cryptographic keys used.

To limit risks to validity, our analysis will make no assumptions about the state of the server beyond what we can glean from the app itself, and we will assume that all interactions, including all cryptographic activities as seen from the device in §4.1.1, are logged to the blockchain, and that these blockchain records are secure, monitored, and immutable. This includes

all ciphertexts in the protocol, as well as any randomness used in the algorithms.

Note that this is an optimistic analysis of the use of the blockchain in this system. It is unlikely that every interaction is stored via the blockchain, and their own documentation of the West Virginia election indicates that the verifying servers are split equally between Amazon AWS and Microsoft's Azure — indicating that their scheme is vulnerable to Microsoft or Amazon surreptitiously adding resources and executing a 51% attack, or performing a selfish mining attack that requires only 1/3 of the compute [26].

Nonetheless, we focus on what is provable given our limited access to the system, and show that this analysis is sufficient to demonstrate a number of significant attacks.

5.1 Client-Side Attacks

We find that an attacker with root privileges on the device can disable all of Voatz's host-based protections, and therefore stealthily control the user's vote, expose her private ballot, and exfiltrate the user's PIN and other data used to authenticate to the server.

5.1.1 Defeating Host-based Malware Detection

The Zimperium SDK included within Voatz is set to detect debugging and other attempts to modify the app, and to collect intelligence on any malware it finds. By default, it would have detected our security analysis, prevented the app from running normally, and alerted the API server of our actions.

As mentioned in §4.4, Zimperium communicates with the Voatz app, and ultimately with Voatz's API server, via a set of callbacks initiated when the app loads. Defeating Zimperium was therefore as simple as overriding its entry points to prevent the SDK from executing. The hooking utilities provided by the Xposed Framework allow us to divert control flow with minimal effort — Figure 8 shows the code to disable one of its two entry points; in total, disabling Zimperium required four lines of code, and is imperceptible to the user.

We assume that there is no out-of-band communication between Zimperium and Voatz, and find no indication in either Zimperium's documentation or in our analysis of the app that this service exists. Additionally, if such communication does exist, it would only marginally increase the effort required to defeat it; one would need to hook other parts of Zimperium that perform detection, or communicate with their server directly.

5.1.2 Full control over the user, on or off device

Once host-based malware detection has been neutralized, an attacker with root privileges has the ability completely control the user, what the user sees, and leak the user's ballot decisions and personal information.

```
argClass = loadClass("com.zimperium.DetectionCallback");

findAndHookMethod("com.zimperium.ZDetection", loader,
  "addDetectionCallback", argClass, new XC_MethodHook() {
    void beforeHookedMethod(MethodHookParam p) {
      p.setResult(null); // prevents method from running
    }
});
```

Figure 8: Code to disable the Zimperium security SDK, slightly simplified

Stealing User Authentication Data: Despite being encrypted with keys that leverage the Android Keystore, the user's PIN and other login information *not* stored in protected storage, and do pass through the application's memory. Exfiltrating these key pieces of information would allow a remote attacker to impersonate the user to Voatz's servers directly, even off-device.

We find that an attacker with root access to the device can surreptitiously steal the PIN and the rest of Voatz's authentication data. In the process of performing our analysis, we developed a tool that intercepts and logs all communication between the device and the server before it is encrypted with SK_{aes}, as well as before data is encrypted and stored in the local database. This allowed us to see, in plaintext, both the user's raw PIN and other authentication data. While our proof of concept stops at logging this information via Android's system debug features (`adb logcat`), it would be trivial broadcast these requests over the network, modify them, or stop them from occurring at all.

An attacker need not necessarily wait until the user decides to vote — offline attacks against Voatz's scheme are also entirely possible. Recall that the database requires only the user's PIN to unlock, and in no way limits the number of times this PIN might be attempted. Worse, the app artificially limits the PIN to exactly 8 numeric characters, meaning that there are only 100,000,000 possible PINs.[17] A brute force attack can therefore easily rediscover the PIN by repeatedly generating keys and attempting to decrypt the database, recovering the PIN, login information, and vote history of the user all at once.[18]

Such a brute force attack can be performed fairly rapidly. Note that an attacker need not do this on-device, as the encrypted database file can be exported. We implemented a prototype of this attack and confirmed that an attacker can brute-force the key in roughly two days on a relatively recent commodity laptop. We conclude that such a threat is viable, particularly if the same installation of Voatz will be used across multiple elections.

[17] Voatz also forbids PINs containing 3 consecutive identical digits, which eliminates ~5% of these.

[18] A salt is also required to unlock the database. This is stored on disk, unencrypted, in the app's shared preferences file.

Stealth UI Modification Attack: It is straightforward to modify the app so that it submits any attacker-desired vote, yet presents the same UI as if the app recorded the user's submitted vote. If the election configuration allows vote-spoiling, there is also a variant of this attack which has been demonstrated on the Estonian e-voting system: allow the user to vote normally, but change the vote once the user closes the app [57].

Similarly, the attacker could stealthily suppress voter's choices if they select an undesired candidate, but continue to show the verification dialog as if the vote had successfully been cast. To the election authority, this might be indistinguishable from the voter failing to submit a ballot. To the voter, this is indistinguishable from correctly voting, at least until the authority releases voter records for that election.[19]

5.2 Server Attacks

We find that, assuming the optimistic use of the blockchain discussed in the threat model, Voatz's server is still capable of surreptitiously violating user privacy, altering the user's vote, and controlling the outcome of the election.

In particular, we find that the protocol discussed in §4.1.1 provides no guarantees against the API server actively altering, viewing, or inventing communication from the device; the server can execute an active MITM attack between the user device and whatever blockchain or mixnet mechanism exists on the other end. Note that there is no other cryptographic operation performed between the device and the server at any point other than the AES encryption, including any sort of cryptographic signing by the device or the device's Keystore. If server performs these cryptographic operations itself — that SK_{aes} is available to the server — it can decrypt the user's ballot before it is submitted to any external log and convincingly re-encrypt any value to be sent to the log.

Even if SK_{aes} is not available to the server — for example, if all cryptographic operations are performed in a Hardware Security Module (HSM) — it must then at least have access to the unencrypted TLS stream, and so it is *still* possible for the server to execute an active MITM attack.

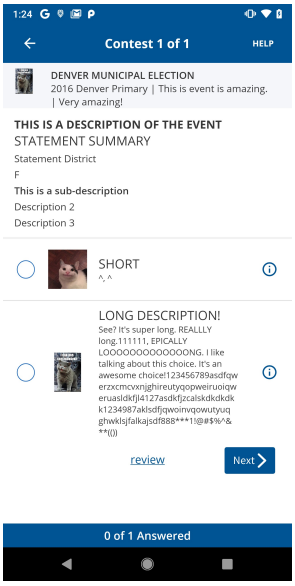
Recall there is no public key authentication performed as a part of the device handshake, and there is no proof or verification by the device that these interactions are ever logged on the blockchain. The server can therefore terminate the connection before the HSM and arbitrarily impersonate the user's device by, e.g., replaying the entire device handshake and all future communication back through the HSM to the blockchain.[20]

[19] In the U.S., public records often list which voters participated in any given election.

[20] Perhaps this hypothetical HSM also contains the TLS keys required to terminate the connection, and performs all cryptographic operations in the enclave. However, all communication is encrypted with SK_{aes}, including those that require queries against databases of users, it is therefore unclear that this is the case, but, even so, the server is capable of performing a number attacks on the user. See §5.3.

Note that, given these attacks, it is unclear if there exists a scheme in which a receipt can convincingly prove that the correct vote was logged.

5.3 Network Adversary



(a) Question.

```
Choice = {
  " choiceDetails ": {"imageUrl":SHORT_IMG,
          "webUrl":SHORT_IMG},
  "choiceId":"1",
  " description ":"Short",
  " description 1":"^",
  " description 2":"^",
  "isWriteIn": False,
  "nonSelectable":False
}
Choice2 = {
  " choiceDetails": { 'imageUrl':
          LONG_IMG_URL,'webUrl' :
          LONG_IMG_URL},
  "choiceId":"2",
  " description ":"Long Description !",
  " description 1":"See? It's super long.
          REALLLY long.111111",
  " description 2":"EPICALLY
          LOOOOOOOOOOOOONG...."
  "isWriteIn": False,
  "nonSelectable":False
}
```

(b) Corresponding JSON for each ballot option.

Figure 9: Voting sidechannel attack explained. All descriptions are generated by our server.

We find that an adversary with the ability to view the user's network activity, without access to *any* key material, can still infer how the user voted. Specifically, in this section we demonstrate that the app leaks the length of the plaintext, which can allow an attacker to learn, at minimum, which candidate the user voted for.

The vulnerability stems from the way in which a ballot is submitted to the server after a user is done selecting their options. As shown in Figure 6, the "choices" list in a vote submission contains only the options selected by the user, and includes with that choice *the entirety of the metadata provided by the server about that candidate*. This, in turn, causes the length of the ciphertext to vary widely depending on the choices of the voter.

Figure 9b shows the differences in metadata sent to and from the server between the two candidates as displayed in-app in Figure 9a. Note that the URLs provided are also potentially variable length, and the length of those URLs is completely imperceptible to the user.

We verified this vulnerability by setting up a proxy between our app and our API server and recording all communication via `tcpdump`. We then used the app to participate in an election twice, once voting for the "short" candidate and once for

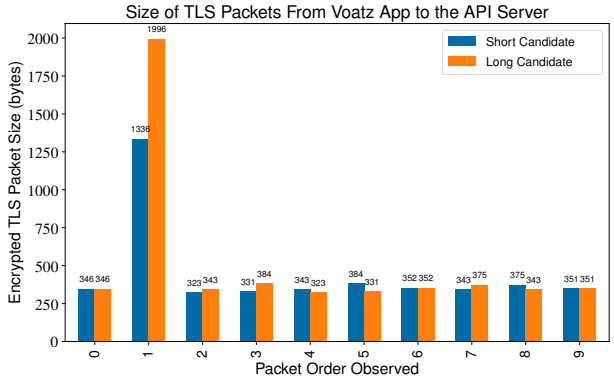


Figure 10: HTTPS encrypted packet lengths immediately after a user submits a vote, in order sent. Note the size of the "short" and "long" candidate in packet 1.

the "long" candidate. Figure 10 shows the resulting ciphertext sizes in bytes (specifically, the TLS Application Data field's length per packet) in both runs — in both cases the second packet (packet #1) corresponds to the actual vote submission, where the rest are other miscellaneous protocol queries involved in vote casting and user maintenance. The length of the encrypted packet clearly leaks which candidate was selected, is easily distinguishable in its size from other packets in the protocol, and, importantly, its size is unaffected by any parameters that vary by user.[21]

It is worth noting that, ironically, Voatz's additional cryptography exacerbates this vulnerability. In typical HTTPS connections, the plaintext is gzip-compressed prior to being encrypted, which offers some privacy in that block-cipher padding is potentially more likely to hide the size differences between plaintexts, or at least obfuscate the plaintext size. Because Voatz encrypts outgoing data *before* it reaches the HTTPS layer, and compressing an encrypted packet will not reduce its size, the compression step is rendered immaterial and the size of the final packet's ciphertext is kept proportional to the size of the plaintext. The result is that (although the figures presented here do intentionally add text to exaggerate the affect for pedagogical purposes), a modest few bytes' difference can be significant.

For this attack to work, we make the following two assumptions:

1. That the attacker can themselves use the app and learn the ballot options presented (perhaps by themselves voting and gaining access to the JSON representation of the ballot options).

2. That the server does not somehow send the ballot options to the device padded to be of equal length.

[21] The size of the ciphertext will not vary depending on the user, but may vary minimally depending on the phone's TLS implementation.

The first assumption is likely not an issue given the attacks presented in §5.1. For example, an attacker need only be a registered voter, or have previously exploited a registered voter's device and witnessed their ballot options, or simply witnessed a voter casting a ballot in a particular way and recording the result.

The second assumption is also a likely non-issue for an attacker, as we find little evidence that the app is defending against this attack — there is no code to remove extraneous symbols or whitespace from ballot questions before they are presented, and other transactions that involve sensitive user information are fully generated device-side and largely independent of the server (like the user's name, age, and location), are also not padded. Finally, even if this assumption does not hold, a limited version of the attack is still viable: if the user votes for *no candidate* and skips the question completely, the device sends the server an empty list.

Note that this sidechannel allows the attacker to detect the voter's intent *before* the ballot arrives at the server. If the attacker is in a position to block packets on their way to the server, (as, for example, an ISP or network owner would), the adversary could intentionally drop this packet and adaptively stop the voter from submitting their ballot. To the user, this would look like a service interruption on Voatz's end, and may degrade the experience enough to stop the voter from casting their ballot at all.

5.4 Other Observations and Weaknesses

Privacy and geostrategic concerns: To sum up the privacy impact of Voatz, information sent to Voatz and/or third parties associated with this service include the user's email, physical address, exact birth date, IP address, a current photo of themselves, their device's model and OS version, and preferred language. Identifying data is provided to Jumio and Voatz, and, to the best of our knowledge, Voatz makes no representations to its users about how long such information is retained, stored, or if it is shared beyond a general privacy policy that does not explicitly discuss Jumio. Worse, if Jumio were to prove truly malicious, it is possible it could refuse to validate particular users at all. Furthermore, we note that the app requests permissions to read the user's GPS upon first login, though we have not identified what exactly the app does with this information.

One of the reported uses of Voatz's software is UOCAVA (overseas military) voters, indicating that information leaked about its users could also potentially provide adversaries with information about US military deployments. Note that, in addition to the PII sent to Voatz and Jumio, the voter's IP address alone can carry information about the user's location, so all third parties may learn roughly where the user is (Jumio, Crashlytics, Zimperium). The result is that these services may learn rough troop movements from their use of the app.

Finally, Voatz makes extensive use of code from a variety of

third party developers. We provide a full listing in Appendix B, and find that the App includes over 22 libraries provided by 20 different vendors.

Risks of Sideloaded Malware & Unsupported Devices Voatz's security requires that the app only be available on certain devices, in particular modern phones with up-to-date operating systems. They implement this via app store preferences; the Google Play Store will only allow certain device models to install the app, and will not make the app visible if the device does not meet Voatz's criteria.

This enables attackers to trick users of unsupported devices into installing an app containing malware by establishing a legitimate-looking website with information about how to vote and directing the reader to install a malicious version of Voatz's app. This is not a hypothetical concern — after the popular game Fortnite was released outside the Play Store to avoid Google's fees, malware authors tricked many naive users using very similar tactics [18].

Susceptibility to Coercion: As mentioned in 4.2, the app never requires the voter to re-enter their PIN at log-in after registration, and does not appear to show the user if a ballot has been re-voted or spoiled.

This indicates that the app leaves users vulnerable to coercion attacks. Consider a voter asleep or otherwise incapacitated. Assuming the attacker has physical access to the device and user, and that the device is unlockable via the user's fingerprint, an attacker would easily have the ability to cast a vote on behalf of the user. This threat model is very relevant in the case of intimate partner abuse [23, 45].

5.5 Voter Verified Receipt

Outside of the password request feature mentioned in §4.3, there is no mention of the receipt in the app, and it does not appear that the app itself provides any method of verifying that the ballot was counted in the blockchain of record — or, beyond Voatz's documentation, that any such blockchain of record exists.

In any event, there are significant practical challenges in providing such receipts. In the case that the app *did* present some sort of concrete cryptographic verification, this could allow the user to prove the way they voted — violating the requirement of a secret ballot and allowing the voter to sell her vote. If the receipt arrives as an encrypted PDF, it is unclear how Voatz can verify to the user that the encrypted PDF actually came from Voatz, and, if it is verified in-app, how one would protect the verification process from the UI modification attacks presented in §5.

Finally, there are significant usability concerns of the receipt that require analysis — What remediation does a user have if the ballot and receipt do not match? How does a user know when to expect a receipt? If the receipt is sent or delayed until post-certification of the election, is there no remediation of a mistake? Transparency in design here would help voters understand these tradeoffs, and without further information, a full analysis of these receipts is not possible.

6 Discussion & Conclusion

Responsible Disclosure: Given the heightened sensitivity surrounding election security issues, and due to concerns of potential retaliation, we chose to alert the Department of Homeland Security (DHS) and anonymously coordinate disclosure through their Cybersecurity and Infrastructure Security Agency (CISA). Before publicly announcing our findings, we received confirmation of the vulnerabilities from the vendor, and, while they dispute the severity of the issues, appear to confirm the existence of the side channel vulnerability, and the PIN entropy issues[22]. We also spoke directly with affected election officials in an effort to reduce the potential for harming any election processes.

The Usefulness of Bug Bounties: As previously mentioned, we use the most recent version of the app as of January 1, 2020. Voatz provides a "bug bounty" version of the app via a third party called HackerOne [4]. We chose not to examine this version of the app for several reasons.

First, choosing to evaluate this bounty app alone would introduce additional threats to validity, and as the differences between this version and the ones that have been fielded are unclear, we chose to err on the side of realism. Note that, being obfuscated in a way that requires lengthy manual processes to undo, the second app would have required nontrivial additional effort to analyze.

Second, crucially, the bounty does not provide any additional helpful insight into Voatz's server infrastructure, nor does it provide any source or binary for the API server to test against. Indeed, when this decision was made, both Voatz's bug bounty app and the Google Play app failed to connect.

Finally, it is unclear if many of the issues we discovered would have been covered by the bug bounty. For example, we assume an attacker with root privileges on-device, and the bug bounty refuses attacks that require physical access. Since our development used manual jail-breaking techniques which require physical access, unlike a nation-state (which is likely to use zero-day vulnerabilities allowing remote exploitation), this restriction would exclude all of our device-side attacks. Further, the bounty puts MITM attacks explicitly out of scope, which would remove the sidechannel attack, or the analysis of the potential attacks from an adversary that controls the API server.

[22] The vendor shared additional information, but as those details were part of confidential communications in the vulnerability disclosure process they are not included in this paper. Nothing provided by the vendor contradicts the factual findings in this paper.

We conclude that the bug bounty is not particularly relevant to improving the security of this system, and serves as an example of how such engagements may not be as effective as one may hope.

A Note on Transparency: Increased transparency would have likely helped Voatz better secure their system. None of the vulnerabilities we discovered are novel; indeed, sidechannel attacks are well known in the cryptographic engineering and in the research literature, and many of the other issues appear to be the result of poor design. Had Voatz been more public about their system, these faults would have been easily recognizable.

It is also clear that Voatz's lack of transparency did not significantly hinder our ability to discover the flaws presented in this paper, and will similarly fail to prevent a well-resourced adversary from doing the same. If anything, an intelligence agency is less likely to take the considerable time and effort we expended to comply with the law. In our analysis, we never intentionally connected to Voatz's servers, and retargeted all communication (including Crashlytics, Jumio, and Voatz's API server) to our own infrastructure. Criminals or foreign intelligence agencies, on the other hand, would likely have no qualms about disrupting normal operations, including by connecting to Voatz's servers or attacking Voatz directly. Such attackers will therefore have an *easier* time discovering exploitable vulnerabilities, and can potentially discover flaws we were unable to find; it is possible that Voatz's backend, server infrastructure, blockchain implementation, and other parts of their service have issues that are impossible to analyze without further access.

Finally, the lack of information available to the user surrounding the use of third party services is ethically dubious. As mentioned in §4.2, the only notification we discovered that indicates to the user that Jumio exists is the logo placed in the lower right corner of the app's photo, and we found no indicator that Zimperium or Crashlytics are being used at all. Further, while the privacy policy does state that "We may transfer Personal Information to third parties for the purpose of providing the Services," it never discloses what information or to whom. A reasonable person may misunderstand and assume that their data, particularly their ID information, is only being shared with Voatz. While it is disclosed in a general sense in the privacy policy, we believe that voting systems, as a crucial civil process, should be held to a much higher standard.

Conclusion: Beginning with West Virginia, Utah, and Colorado, the US has ventured down the path of Internet voting. Despite the concern expressed by experts, one company has sold the promise of secure mobile voting, using biometrics, blockchain, and hardware-backed cryptography.

Yet our analysis has shown that this application is not secure. A passive network adversary can discover a user's vote, and an active one can disrupt transmission in response. An attacker that controls a user's device also controls their vote, easily brushing aside the app's built-in countermeasures. And our analysis of the protocol shows that one who controls the server likely has full power to observe, alter, and add votes as they please.

A natural question may be why such a service was fielded in the first place. Speaking to the Harvard Business Review, Voatz backer and political philanthropist Bradley Tusk stated:

> *It's not that the cybersecurity people are bad people per se. I think it's that they are solving for one situation, and I am solving for another. They want zero technology risk in any way, shape, or form. [...] But in my view, then you can't resolve the issues on guns, on climate, on immigration, because the middle 70% doesn't participate in primaries [...] I am solving for the problem of turnout. [61]*

While we appreciate and share Tusk's desire to increase voter participation, we do not agree that the security risks in this domain are negligible; we believe that the issues presented in this work outweigh the potential gains in turnout.[23] As we have shown in this paper, vulnerabilities in Voatz and the problems caused by a lack of transparency are very real; the choice here is not about turnout, but about an adversary controlling the election result and a loss of voter privacy, impugning the integrity of the election.

Given the severity of failings discussed in this paper, the lack of transparency, the risks to voter privacy, and the trivial nature of the attacks, we suggest that any near-future plans to use this app for high-stakes elections be abandoned. We further recommend that any future designs for voting systems (and related systems such as e-pollbooks) be made public, and that their details, source, threat model, as well as social and human processes be available for public scrutiny.

Note that all attacks presented in this paper are viable regardless of the app's purported use of a blockchain, biometrics, hardware-backed enclaves, and mixnets. We join other researchers in remaining skeptical of the security provided by blockchain-based solutions to voting [37, 50, 51], and believe that this serves as an object lesson in security — that the purported use of a series of tools does not indicate that a solution provides any real guarantees of security.

It remains unclear if *any* electronic-only mobile or Internet voting system can practically overcome the stringent security requirements on election systems. Indeed, this work adds to the litany of serious flaws discovered in electronic-only approaches, and supports the conclusion that the current standard — software independent [52] systems using voter-verified paper ballots and Risk Limiting Audits [44] — remain

[23] Indeed, it is unclear if mobile and internet voting actually increases voter turnout. A study from Switzerland [31] finds, somewhat surprisingly, no statistically significant increase in voter participation over more traditional forms of balloting.

the most secure option. It is the burden of the developer of any system to prove that their system is as secure, to both the public and the security community, before it can be trusted as a crucial component in the democratic process.

Acknowledgments

For guiding us through the responsible disclosure process, we are eternally thankful for the team at the BU/MIT Technology Law Clinic led by Andy Sellars, Tiffany C. Li, and students John Dugger, Quinn Heath, and Eric Pfauth. Without this fantastic team's advice, patience, and effort, this paper would never have been released. We would further like to thank Matt Blaze, Matt Green, Neha Narula, Sunoo Park, Ron Rivest, and Gerry Sussman for providing feedback and insight that helped form this paper.

Michael Specter and Danny Weitzner are supported, in part, by the MIT Internet Policy Research Initiative, and Specter is further supported by the Google's Android Security and PrIvacy REsearch (ASPIRE) fellowship. James Koppel was supported by Toyota Research Institute.

References

[1] Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps. https://ibotpeaches.github.io/Apktool/.

[2] Magisk Manager For Android 2019. https://magiskmanager.com/.

[3] Realm: Create reactive mobile apps in a fraction of the time. https://realm.io/.

[4] Voatz - Bug Bounty Program. https://hackerone.com/voatz.

[5] Voatz FAQ. https://voatz.com/faq.html [https://perma.cc/FBQ8-N875].

[6] Stay secure with jumio's certified 3d liveness detection s. https://www.jumio.com/about/press-releases/3d-liveness-detection/, 2018.

[7] Ben Adida. Helios: Web-based Open-Audit Voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.

[8] Andrew Yang. Modernize Voting. https://www.yang2020.com/policies/modernize-voting/.

[9] Android. Android keystore system. https://developer.android.com/training/articles/keystore.

[10] Android. Settings.Secure | Android Developers. https://developer.android.com/reference/android/provider/Settings.Secure.

[11] Kevin Beaumont. Somebody sent me a link to another github account, with the author name listed at voatz. it has hardcoded username and passwords. https://twitter.com/GossiTheDog/status/1026904510386585600, August 2018.

[12] Kevin Beaumont. The voatz website is running on a box with out of date ssh, apache (multiple cvss 9+), php etc. https://twitter.com/GossiTheDog/status/1026607447996354561, August 2018.

[13] Susan Bell, Josh Benaloh, Michael D Byrne, Dana DeBeauvoir, Bryce Eakin, Philip Kortum, Neal McBurnett, Olivier Pereira, Philip B Stark, Dan S Wallach, et al. Star-vote: A secure, transparent, auditable, and reliable voting system. In *2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 13)*, 2013.

[14] Josh Benaloh. Simple Verifiable Elections. *EVT*, 6:5–5, 2006.

[15] Josh Benaloh. Ballot Casting Assurance via Voter-Initiated Poll Station Auditing. *EVT*, 7:14–14, 2007.

[16] Matthew Bernhard, Josh Benaloh, J. Alex Halderman, Ronald L. Rivest, Peter YA Ryan, Philip B. Stark, Vanessa Teague, Poorvi L. Vora, and Dan S. Wallach. Public evidence from secret ballots. In *International Joint Conference on Electronic Voting*, pages 84–109. Springer, 2017.

[17] Matt Blaze, Jake Braun, and Cambridge Global Advisors. DEFCON 25 Voting Machine Hacking Village. *Proceedings of DEFCON, Washington DC*, pages 1–18, 2017.

[18] Brian Barrett. Impostor fortnite android apps are already spreading malware. https://www.wired.com/story/imposter-fortnite-android-apps-already-spreading-malware/, 2018.

[19] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L Rivest, Emily Shen, et al. Scantegrity ii municipal election at takoma park: The first e2e binding governmental election with ballot privacy. 2010.

[20] D. Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security Privacy*, 2(1):38–47, January 2004.

[21] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L Rivest, Peter YA Ryan, Emily Shen, and Alan T Sherman. Scantegrity ii: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. *EVT*, 8:1–13, 2008.

[22] Connie Loizos. Voatz has raised $7 million in Series A funding for its mobile voting technology, June 2019. http://social.techcrunch.com/2019/06/06/voatz/.

[23] Sunny Consolvo. Privacy and Security Practices of Individuals Coping with Intimate Partner Abuse. 2017.

[24] David Jefferson, Duncan Buell, Kevin Skoglund, Joe Kiniry, and Joshua Greenbaum. What We Don't Know About the Voatz "Blockchain" Internet Voting System. https://cse.sc.edu/~buell/blockchain-papers/documents/WhatWeDontKnowAbouttheVoatz_Blockchain_.pdf, May 2019.

[25] Election Assistance Commission. EAC Releases Annual Grant Expenditure Report, August 2017. https://www.eac.gov/news/2017/08/16/eac-releases-annual-grant-expenditure-report.

[26] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.

[27] Ariel J. Feldman, J. Alex Halderman, and Edward W. Felten. Security analysis of the Diebold AccuVote-TS voting machine. 2006.

[28] David Fifield. A better zip bomb. In *13th USENIX Workshop on Offensive Technologies (WOOT 19)*, 2019.

[29] Forrest Centi. The denver mobile voting pilot: A report. https://cyber-center.org/wp-content/uploads/2019/08/Mobile-Voting-Audit-Report-on-the-Denver-County-Pilots-FINAL.pdf, 2018.

[30] Pierrick Gaudry. Breaking the encryption scheme of the Moscow internet voting system. *arXiv preprint arXiv:1908.05127*, 2019. https://arxiv.org/pdf/1908.05127.pdf.

[31] Micha Germann and Uwe Serdült. Internet voting and turnout: Evidence from switzerland. *Electoral Studies*, 47:1–12, 2017.

[32] Dan Goodin. FBI tells router users to reboot now to kill malware infecting 500k devices. https://arstechnica.com/information-technology/2018/05/fbi-tells-router-users-to-reboot-now-to-kill-malware-infecting-500k-devices/, May 2018.

[33] Dan Goodin. Mass router hack exposes millions of devices to potent NSA exploit, November 2018. https://arstechnica.com/information-technology/2018/11/mass-router-hack-exposes-millions-of-devices-to-potent-nsa-exploit/.

[34] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *41st IEEE Symposium on Security and Privacy*, 2019.

[35] Harvie Branscomb. Denver voatz on cell phones – initial review. http://electionquality.com/2019/05/denver-voatz-1/, May 2019.

[36] Jed Pressgrove. The Hack Attempt Against Voatz 'Not Close,' Officials Say. https://www.govtech.com/security/The-Hack-Attempt-Against-Voatz-Not-Close-Officials-Say.html, October 2019.

[37] Jefferson, David et al. What we don't know about the voatz "blockchain" internet voting system. https://cse.sc.edu/~buell/blockchain-papers/documents/WhatWeDontKnowAbouttheVoatz_Blockchain_.pdf, May 2019.

[38] Jen Kirby. West Virginia is testing a mobile voting app for the midterms. What could go wrong? https://www.vox.com/2018/8/17/17661876/west-virginia-voatz-voting-app-election-security, August 2018.

[39] Douglas Jones and Barbara Simons. *Broken ballots: Will your vote count?* CSLI Publications Stanford, 2012.

[40] Kevin Collier. FBI is investigating alleged hacking attempt into mobile voting app. https://www.cnn.com/2019/10/01/politics/fbi-hacking-attempt-alleged-mobile-voting-app-voatz/index.html, October 2019.

[41] Amy Klobuchar. S.1540 - 116th Congress (2019-2020): Election Security Act of 2019, May 2019. https://www.congress.gov/bill/116th-congress/senate-bill/1540/text.

[42] Larry Moore and Nimit Sawhney. UNDER THE HOOD The West Virginia Mobile Voting Pilot, 2019. https://www.nass.org/sites/default/files/2019-02/white-paper-voatz-nass-winter19.pdf.

[43] Liat Weinstein. University of michigan students implicated in potential voting app hack. https://www.michigandaily.com/section/news-

briefs/university-michigan-students-implicated-potential-voting-app-hack, 2019.

[44] Mark Lindeman and Philip B. Stark. A gentle introduction to risk-limiting audits. *IEEE Security & Privacy*, 10(5):42–49, 2012.

[45] Tara Matthews, Kathleen O'Leary, Anna Turner, Manya Sleeper, Jill Palzkill Woelfer, Martin Shelton, Cori Manthorne, Elizabeth F. Churchill, and Sunny Consolvo. Stories from survivors: Privacy & security practices when coping with intimate partner abuse. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2189–2201. ACM, 2017.

[46] Maya Kosoff. "a horrifically bad idea": Smartphone voting is coming, just in time for the midterms. https://www.vanityfair.com/news/2018/08/smartphone-voting-is-coming-just-in-time-for-midterms-voatz, 2018.

[47] Lucas Mearian. Why blockchain-based voting could threaten democracy. *Computerworld*, August 2019. https://www.computerworld.com/article/3430697/why-blockchain-could-be-a-threat-to-democracy.html.

[48] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[49] Robert W. Ney. H.R.3295 - 107th Congress (2001-2002): Help America Vote Act of 2002, October 2002. https://www.congress.gov/bill/107th-congress/house-bill/3295.

[50] Sunoo Park, Michael Specter, Neha Narula, and Ronald L. Rivest. Going from bad to worse: from internet voting to blockchain voting. (DRAFT).

[51] J. Alex Halderman Rachel Goodman. Internet voting is happening now and it could destroy our elections., January 2020. https://slate.com/technology/2020/01/internet-voting-could-destroy-our-elections.html.

[52] Ronald L. Rivest. On the notion of 'software independence'in voting systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3759–3767, 2008.

[53] David G. Robinson and J. Alex Halderman. Ethical issues in e-voting security analysis. In *International Conference on Financial Cryptography and Data Security*, pages 119–130. Springer, 2011.

[54] Ron Wyden. Sen. Ron Wyden (D-Ore.) Letter Regarding Voatz. https://www.washingtonpost.com/context/sen-ron-wyden-d-ore-letter-regarding-voatz/e9e6dd4f-1752-4c46-8e37-08a0f21dd042/, November 2019.

[55] Peter YA Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. Prêt à voter: a voter-verifiable voting system. *IEEE transactions on information forensics and security*, 4(4):662–673, 2009.

[56] skylot. skylot/jadx, January 2020. https://github.com/skylot/jadx.

[57] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. Security analysis of the Estonian internet voting system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 703–715. ACM, 2014.

[58] Steven Rosenfeld. Counting voatz: Inside america's most radical voting technology. https://www.nationalmemo.com/counting-voatz-inside-americas-most-radical-voting-technology/, May 2019.

[59] Bennie G. Thompson. H.R.2660 - 116th Congress (2019-2020): Election Security Act of 2019, June 2019. https://www.congress.gov/bill/116th-congress/house-bill/2660/text.

[60] Voatz. Statement on Sen. Wyden's Letter, November 2019. https://blog.voatz.com/?p=1133.

[61] Mitchell Weiss and Maddy Halyard. *Voatz*. Harvard Business Review, 2019. Case Study.

[62] Scott Wolchok, Eric Wustrow, Dawn Isabel, and J. Alex Halderman. Attacking the Washington, DC Internet voting system. In *International Conference on Financial Cryptography and Data Security*, pages 114–128. Springer, 2012.

[63] Yael Grauer. Safe Harbor, or Thrown to the Sharks by Voatz? https://magazine.cointelegraph.com/2020/02/07/safe-harbor-or-thrown-to-the-sharks-by-voatz/, February 2020.

[64] Kim Zetter. Virginia Finally Drops America's 'Worst Voting Machines'. *Wired*, August 2015. https://www.wired.com/2015/08/virginia-finally-drops-americas-worst-voting-machines/.

A Annotated Security Claims

As we have mentioned in various parts of this paper, Voatz has presented no formal threat model, and has failed to release a full description of their system. Instead, Voatz makes a number of claims via their FAQ [5] about their security guarantees.

We felt that it was important to discuss these in attempting to better understand the intended threat model. Below are quotes directly from the FAQ at the time of writing, and our analysis of these claims given the exploration in this paper.

> *First, only recently-manufactured smartphone models from Apple, Samsung and Google are supported with Voatz. These devices are built with security features, like fingerprint and facial recognition, that extend far beyond standard browsers running on a potentially-compromised PC for voter authentication Second, modern smartphones provide hardware-based security to store private keys which, in turn, allow highly secure, encrypted transactions to be conducted over the public Internet.*

While Voatz does use the Android Keystore to encrypt the PIN, the PIN itself is not stored in a hardware enclave. It is therefore possible for an adversary with root access to the device to exfiltrate the PIN and all data protected by the PIN, as mentioned in §5.1.2.

> *The Voatz app takes advantage of the capabilities of the supported smartphones, which can detect if the operating system has been tampered with (e.g. an operation known as a "jailbreak"). The Voatz app does not permit a voter to vote if the operating system has been compromised.*

Although the app does attempt to detect such jailbreaks, we believe that the app is unable to prevent targeted attacks that disable Zimperium §5.1.1. Other attempts to detect jailbreaks (e.g. Google's safetynet) are also commonly defeated via readily available tools like `magiskHide`. Further, it is worth mentioning that device exploitation is not equivalent to jailbreaking the device; an attacker can gain root privileges via, say, an exploit into the Android kernel without persistence.

> *If a device is compromised, the Voatz platform goes to significant lengths to prevent a vote from being submitted. Beyond only operating with certain classes of smartphones with the latest security features, Voatz ensures end-to-end vote encryption and uses multiple approaches for malware detection.*

Voatz implements its device-restriction via filtering on the Google Play Store. While this prevents honest users from voting an outdated devices, there is no barrier to an attacker installing it on any chosen device. The app does check whether it was installed from the Play store or sideloaded, but this is trivial to defeat by text-editing the app's shared preferences. We hence experienced little barrier to running and attacking the app on an unsupported Xiaomi Mi 4i.

As far as we have been able to discern, the Voatz app uses only one malware detection utility – Zimperium. It is possible the server is running some sort of malware detection as well, though we are unable to confirm this.

Finally, it is unclear why end-to-end encryption is mentioned. First, a natural question is what the vendor considers the "end" at which a vote can be unencrypted. Second, if the device is considered an "end" here, it is unclear how this encryption helps if the device performing the cryptography is infected.

> *The Voatz app is built with security measures embedded in qualified smartphones and employs blockchain technology to ensure that, once submitted, votes are verified and immutably stored on multiple, geographically diverse verifying servers*

We have no visibility into the Voatz backend, so it's possible that a blockchain is used in vote storage. However, we found no reference to the blockchain within the app itself. Further, given the attacks we have outlined, it is unclear if the immutability provided by the blockchain is particularly helpful.

> *In addition, Voatz generates a voter-verified audit trail with each vote cast. Upon casting a vote, voters also receive an automatic, digitally-signed receipt with their selections in order to review that their vote was recorded properly. The election organizer also receives an anonymized copy of the digital receipt, ensuring that a post-election audit may be conducted between the paper trail, the anonymized receipt, and the blockchain.*

Voter verifiable audit trails are commonly defined in the literature as the voter being able to see the ballot of record – that is, a ballot that is actually counted by the final tally. We find no indication that voters are able to query the blockchain (or see proofs of inclusion) directly to confirm that their vote was recorded.

> *Beyond enabling multiple audit trails, Voatz has submitted the smartphone voting app to independent third party security firms for audit and undergoes frequent, rigorous, ongoing "red-team" testing. In addition, Voatz is the 1st elections company in the world to run an open bug bounty program on HackerOne for community vetting of its upcoming platform releases.*

At the time of our writing, and to the best of our knowledge, no other security audit appears to have been released publicly. Further, the bug bounty from HackerOne states that "Attacks requiring MITM or physical access to a user's device" are out of scope. Note that these could be construed to include many of the attacks we discovered, and certainly includes the sidechannel attack.

Once submitted, all information is anonymized, routed via a "mixnet" and posted to the blockchain.

We cannot confirm, or find any evidence of, the anonymity guarantees claimed here. We can report that the Voatz app initially transmits vote information by HTTPS to `voatzapi.nimsim.com`, via `GET` and `POST` requests.

Yes, a paper ballot is generated on election night for every mobile vote recorded on the blockchain and the printed ballots are tallied using the standard counting process at each participating county. This also facilitates a post-election audit by comparing the paper ballots with the anonymized voter-verified digital receipts generated at the time of vote submission.

See voter verifiable ballot of record discussion above.

Once the voter is verified, election jurisdictions initiate the process by sending a qualified voter a mobile ballot. Contained in the mobile ballot are "tokens" — think of them as potential votes — which are cryptographically tied to a candidate or ballot measure question. The number of tokens a given voter receives is the same as the number of ovals he or she would have received on a paper ballot handed out at the voter's precinct or sent through the mail. The voter then makes their desired selections on the Voatz app on their smartphone. These selections alter the tokens (like filling in a ballot oval). Overvotes are prevented, as each voter only receives a total number of tokens as they have potential votes. Once submitted, the votes for choices on the ballot are verified by multiple distributed servers called "verifiers", or validating nodes. Upon verification, the token is debited (i.e. subtracted) from the voter's ledger and credited (i.e. added) to the candidate's ledger. The blockchain on every verifier is automatically updated and the process repeats as additional voters submit their selection

We again find little evidence of the use of crypto tokens or the blockchain. Such an algorithm could be done server-side, but it is unclear if this is at all relevant to the security of the system.

Blockchain technology, when used for financial transactions like Bitcoin, cannot be totally anonymous (rendering the term "pseudonymous"), however, when used in voting with the Voatz application, the identity of the voter is doubly anonymized: first by the smartphone, and second by the blockchain server network.

A key question here is "pseudonymous with respect to what adversary?" We find that both Voatz's server and Jumio receive quite a bit of personally identifiable information about the user, and a the API server can clearly link the voter and their ballot if it proved to be malicious. Hopefully Voatz and Jumio delete and do not use such information, but this claim is very unclear.

Detecting a compromised mobile network is particularly challenging for a mobile application, which is why ensuring end-to-end vote encryption and vetting the certificates represented by unique IDs stored on the smartphone are two of the approaches we use to mitigate a compromised mobile network.

We address the end-to-end confusion above, but it is worth noting that A. The unique IDs stored on-device are not certificates in the cryptographic sense, and B. nothing in this paragraph could be construed as preventing the sidechannel attack presented in §5.

B List of Third Parties Used

Voatz makes extensive use of third-party libraries from at least 20 different vendors. We have not confirmed that all of these libraries are actively used by the app, but that they are included in code. Further, a large swath of Voatz's code is obfuscated, so there may be further libraries used that we are unaware of.

- Jumio
- Zimperium
- Amazon AWS
- Realm DB
- Google's Firebase / Crashlytics, gson, protobufs, zxking libraries
- Datatheorem's TrustKit Android
- Facebook's SoLoader (https://github.com/facebook/SoLoader) and fresco (https://github.com/facebook/fresco)
- Keepsafe's relinker (https://github.com/KeepSafe/ReLinker)
- Samsung's knox libraries
- Microblink's data capture libraries
- Takisoft's Preference Manager (https://github.com/takisoft/preferencex-android)
- MichaelRocks libphonenumber (https://github.com/MichaelRocks/libphonenumber-android)

- ReactiveX http://reactivex.io/
- Relex CircleIndicator https://github.com/ongakuer/CircleIndicator
- zhanghai material progressbar https://github.com/zhanghai/MaterialProgressBar
- Square OkHTTP https://square.github.io/okhttp/ and Retrofit https://square.github.io/retrofit/
- JetBrains Anko https://github.com/Kotlin/anko
- Joda.time https://www.joda.org/joda-time/
- Jake Wharton's Timber logging https://github.com/JakeWharton/timber
- ChrisJenX's calligraphy font libraries https://github.com/chrisjenx/Calligraphy