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, James Koppel, and Daniel Weitzner, *MIT*

https://www.usenix.org/conference/usenixsecurity20/presentation/specter

This paper is included in the Proceedings of the 29th USENIX Security Symposium.

August 12–14, 2020

978-1-939133-17-5

Open access to the Proceedings of the 29th USENIX Security Symposium is sponsored by USENIX.

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. As a result of our work, one county in Washington has already aborted their use of Voatz in the 2020 primaries.

1 Introduction

In 2018, Voatz, a private Boston-based company, made history by fielding the first Internet voting app used in high-stakes [1] 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 [45]. The company has recently closed a $7-million series A [27], 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 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) [59], a bill targeted toward helping states move away from outdated and problematic punchcard-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 [31].

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 are extremely vulnerable to a wide range of attacks, allowing adversaries to surreptitiously change the outcome of an election [21, 22, 33, 49, 77].

Today, we are witnessing similar developments in response

*With appreciation to Barbara Simons [46]

†EECS PhD Candidate, CSAIL, Internet Policy Research Initiative

‡EECS PhD Candidate, CSAIL, Computer Assisted Programming Group

§Research Scientist, CSAIL, Internet Policy Research Initiative

[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. [10]).

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. Viability of these attacks may be dependent on the configuration of the particular election, (the ballot style, metadata, etc.), see the relevant section listed for explicit details.

to Russia's interference in the 2016 U.S. Presidential election. Bills have been introduced in both the U.S. Senate [48] and House [70] 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 as the blockchain [57], 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 [56], and one 2020 Democratic presidential candidate has included voting from a mobile device via the blockchain in his policy plank [11]. To our knowledge, only Voatz has successfully fielded such a system.

Unfortunately, the public information about Voatz's system is incomplete. Voatz's FAQ [6], blog, and white paper [50] 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 [29, 60], as well as elected representatives [63], Voatz has declined to provide formal details, citing the need to protect their intellectual property [71]. 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 [44, 47, 51, 75].

This opaque stance is a threat to the integrity of the electoral process. Given the contentious nature of high-stakes elections, 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. In any democracy, 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 [42], Moscow [37], Estonia [68], and Washington D.C. [74]), 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).

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 discuss 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. 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 [19].

Receipt Freeness, Privacy, & Coercion Resistance: Secret-ballot voting systems need to ensure that 1) No voter is able to prove their selections (*Receipt-Freeness*), 2) that no voter's choices can be surreptitiously released or inferred (*Privacy*), and 3) that a voter cannot cooperate with a coercer to prove the way they voted (*Coercion Resistance*). These properties are required 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 or forced to be revealed, they may suffer harassment and coercion [20, 30].

End-to-End Verifiability: End-to-End Verifiable (E2E-V) 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 coercion resistance, privacy, and receipt freeness using techniques such as visual cryptography, homomorphic cryptography, invisible ink, and mixnets [17, 23, 25, 65].

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 [6].

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.

End-to-End vote encryption: Voatz makes multiple references to votes themselves being encrypted "end to end." To the authors' 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 an aggregate vote, maintaining individual voter privacy [18], 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."

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."

Voter Verified Audit Trail: Voatz claims that voters receive a cryptographically-signed digital receipt of their ballot after their vote has been submitted. The guarantees of such a receipt are unclear, although, perhaps this is meant to provide similar guarantees as E2E-V cryptosystems.

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 [29] 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 [43, 69]. Kevin Beaumont found what appeared to be several Voatz service-related credentials on a public Github account [14], and that the Voatz webserver was running several unpatched services [15]. Voatz responded citing a report from the Qualys SSL checker as evidence of the site's security [55], and later claimed that the insecure server Beaumont identified was an intentionally-insecure "honeypot operation" [73]. 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 [63].

3 Experimental Methodology

As performing a security analysis against a running election server would raise a number of unacceptable legal and ethical concerns [62], we instead chose to perform all of our analyses in a "cleanroom" environment, connecting only to our own servers. Special care was taken to ensure that our static and dynamic analysis techniques could never interfere with Voatz or any related services, and we went through great effort so that nothing was intentionally transmitted to Voatz's servers.[2]

To gain a better understanding of Voatz's infrastructure, we began by decompiling the most recent version of their Android[3] application as found on the Google Play Store as of January 1, 2020[4] and iteratively re-implemented a minimal server that performs election processes as visible from the app itself. This included interactions involved in device

[2] Indeed, at the time of analysis, Voatz's servers appeared to be down when tested with an unmodified app on a supported and up-to-date device.

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

[4] Version 1.1.60, SHA256 191927a013f6aae094c86392db4ecca825866ae62c6178589c02932563d142c1

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 to 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 using an Android modding framework. 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.

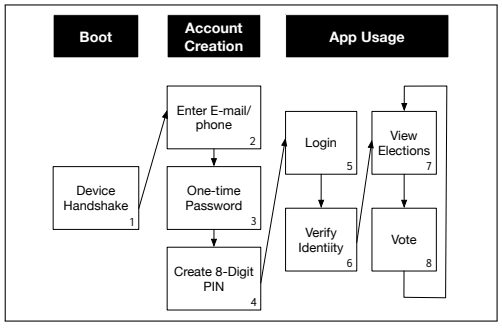


Figure 1: Voatz's workflow as seen from the device.

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.

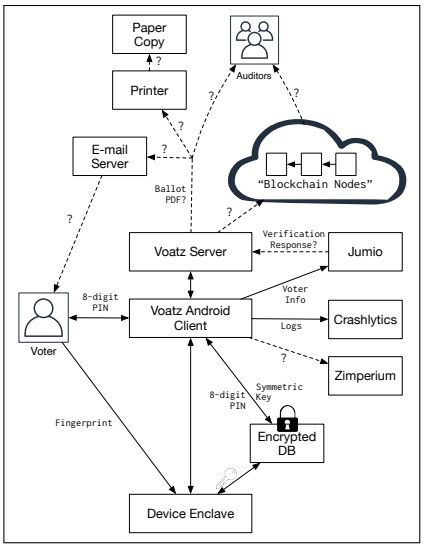


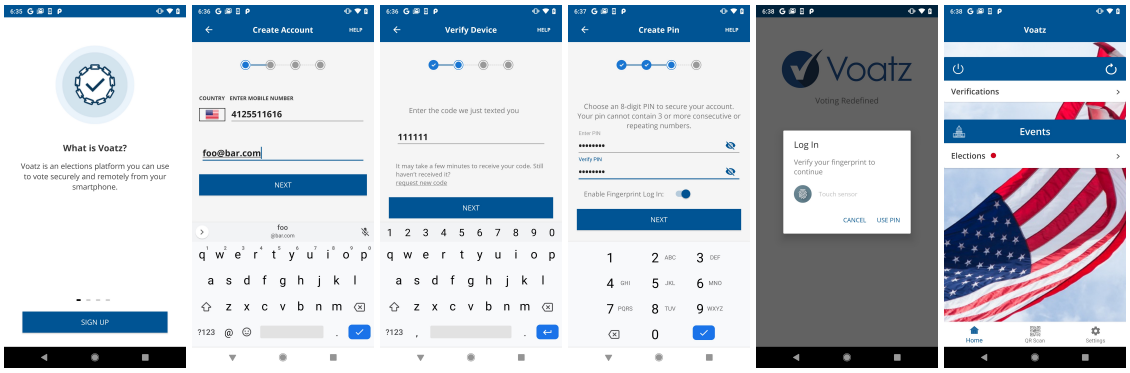
Figure 2: Dataflow between Voatz components and external services. Dashed lines are believed to exist but have not been directly observed.

4.1 Process Overview

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

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 §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.
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,"[5] so

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



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

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

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 [50] and by the Denver Elections Division [35].

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).

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 keypairs, 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 and ECDH-Encrypt$(SK_{ecdh}, SK_{aes}||N||T)$

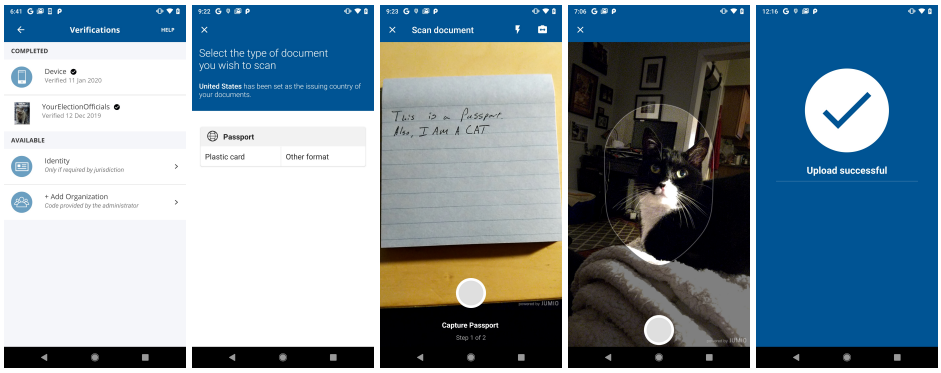
5. Out of the 100 public keys sent by the Server, the App selects the 57th pubkey (PK_S), and finishes the ECDH handshake to create the ECDH shared key SK_{ecdh}. Finally, it decrypts and parses the AES-GCM parameters(SK_{aes}, N, T).

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 an 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



(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.

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.

The Android Keystore is a system service that, if used correctly,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.[6] 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,[7] 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). 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).[8] Finally, after receiving a 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, and the only way a user would be aware that this data is 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 events' ballots, each ballot's particular questions, and the options available for those questions.

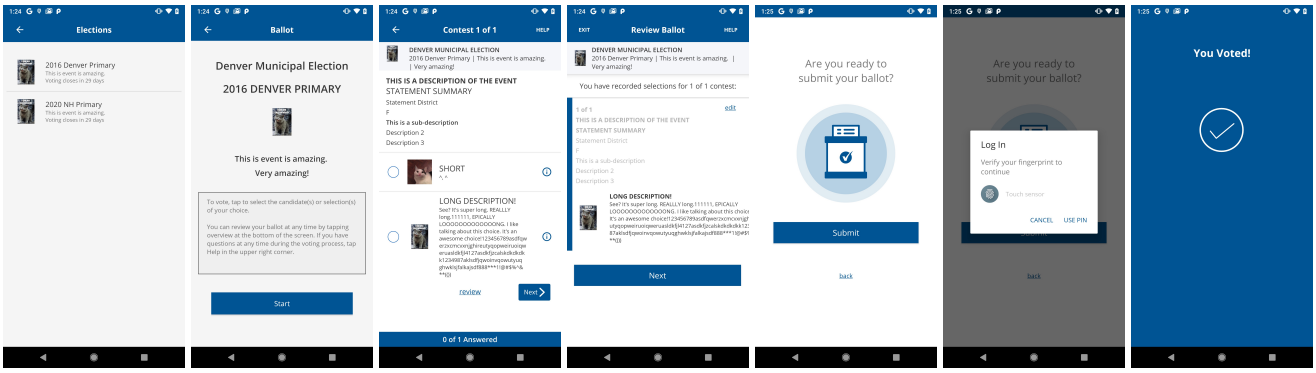
The voter begins by selecting an event (5a), and is then able to view questions associated with these particular events, select responses (or no response at all, depending on the event configuration), and submit their response to the server. At the point of submission, the user is again asked to decrypt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

[6] See Android's Keystore documentation for details [12].

[7] See [13] for more information about Android's local device UUIDs.

[8] Furthermore, Jumio itself has disclosed that it uses a third party, Facetec, to help analyze the video selfies [7]. 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.

is the "auditToken", but this string is never altered by the app, and appears to be a single, static value. Figure 10 shows the entirety of what is sent to the server, AES-GCM encrypted, after a user submits their vote.

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 on-device encrypted database to protect locally stored sensitive data.

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 locally stored in an encrypted database using the Realm database framework [4]. 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 [66]. 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 [34], 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 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, see §5.1.1.

Jumio voter confirmation: Jumio's documentation discusses at length the optional ability to communicate with Jumio's servers for out-of-band verification of a user. Since this is well a documented feature of the system, we assume that Voatz's API server receives confirmation directly from Jumio's servers 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 [35]. Footage of the audit of the 2019 Denver Municipal elections shows that the auditing process consists of manually inspecting 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 [43].

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 the server-provided password (see Figure 6). 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 6.

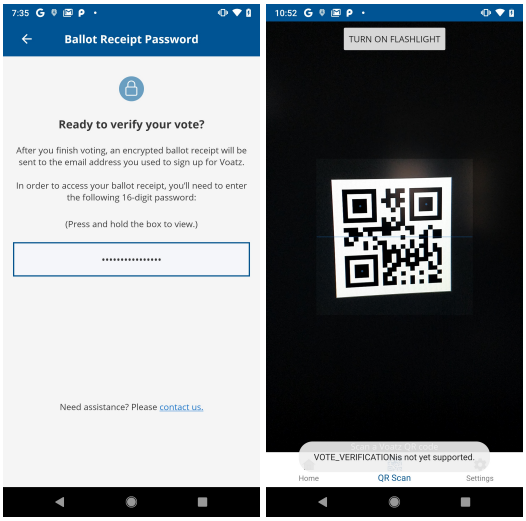


Figure 6: Left: the password request screen. Right: the QR code capture screen; note the popup indicating that the VOTE_VERIFICATION QR code type is unimplemented.

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 control of a user's device,
2. An attacker that has control over Voatz's API server, and
3. A network adversary that can intercept network activity between voter's device and the API server, but has no further access.

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 [39, 40]), 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 trust the system, rather than 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 Voatz's 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 power [32].

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 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.

```
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 7: Simplified code to disable the Zimperium security SDK.

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 7 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. 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 to completely control the user's actions and view of the app, as well as leak the user's ballot decisions and personal information.

Stealing User Authentication Data: Despite being encrypted with keys that leverage the Android Keystore, the user's PIN and other login information are *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 to 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.[9] 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.[10]

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 3.1GHz 2017 MacBook Pro. We conclude that such a threat is viable, particularly if the same installation of Voatz will be used across multiple elections.

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 submission. If the election configuration allows vote-spoiling, there is also a variant of this attack previously demonstrated on the Estonian e-voting system: allow the user to vote normally, but change the vote once the user closes the app [68].

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.[11]

5.2 Server Attacks

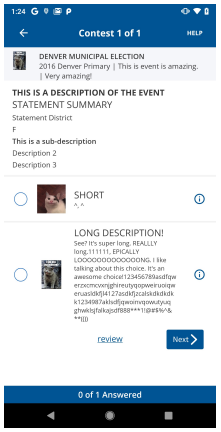
We find that, assuming the optimistic use of the blockchain discussed in the threat model, Voatz's server is still capable of surreptitiously violating user privacy, altering the user's vote, and controlling the outcome of the election.

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 the 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.[12] 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
                LOOOOOOOOOOOONG...."
    " isWriteIn ": False,
    " nonSelectable ":False
}
```

(b) Corresponding JSON.

Figure 8: Voting sidechannel attack explained.

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

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

[11] For U.S. elections, public records often list which voters participated.

[12] 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 of attacks on the user. See §5.3.

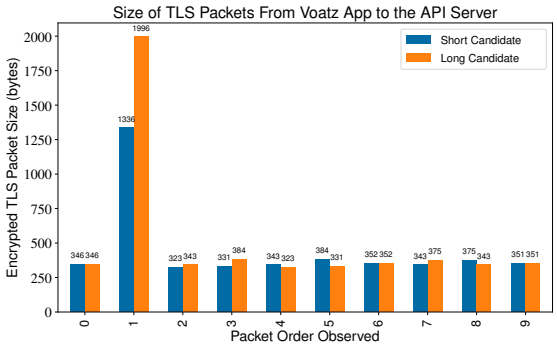


Figure 9: TLS encrypted packet lengths immediately after a user submits a vote, in order sent. Note the size of the "short" and "long" candidate in packet 1.

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 10, 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 8b shows the differences in metadata sent to and from the server between the two candidates as displayed in-app in Figure 8a. Note that the URLs and other metadata provided are also potentially variable length, and the length of the URL 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 the "long" candidate. Figure 9 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 this packet clearly leaks which candidate was selected, is easily distinguishable from other packets in the protocol, and, importantly, its size is unaffected by any parameters that vary by user.[13]

It is worth noting that, ironically, Voatz's additional cryptography exacerbates this vulnerability. In Voatz's implementation, data is gzip-compressed at the application layer prior to being encrypted via TLS, which could have offered some privacy, assuming the compression alone was enough to hide the size differences between plaintexts. Because Voatz encrypts outgoing data *before* the system applies gzip, and compressing an already encrypted payload will not reduce its size, this step is rendered immaterial and the length 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 enough to determine the voter's preferences.

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

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

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

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, have previously exploited a registered voter's device and witnessed their ballot options, or otherwise monitored a voter casting a ballot in a particular way and recorded the result.

The second assumption is also a likely to hold, as we find no 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 independent of the server (like the user's name, age, and location), and are also not padded. Finally, if this assumption does not hold, a limited version of the attack is still viable: if the user selects *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: The Voatz app is incredibly privacy invasive. Information sent to Voatz and/or third parties associated with this service include the user's

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

email, physical address, exact birth date, IP address, a current photo of themselves, their device's model and OS version, and preferred language. The app also requests permissions to read the user's GPS upon first login, though we have not identified what exactly the app does with this information. Finally, Voatz makes extensive use of third party code (see Appendix B); Voatz includes over 22 libraries provided by 20 different vendors.

One of the reported uses of Voatz's software is overseas military voters, indicating that information leaked about its users could also potentially provide adversaries with information about U.S. military deployments. Note that the voter's IP address alone can carry information about the user's location — so Jumio, Crashlytics, and Zimperium can therefore infer troop deployments.

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 [28, 54].

5.5 Voter Verified Receipt

From what can be discerned from the available documentation and the app's code, it is very unclear what guarantees Voatz's receipt provides. Outside of the password request feature mentioned in §4.3, there is no mention of the receipt in the app or its binary, and it does not appear that the app provides any method of verifying that the ballot was counted in the blockchain of record — or, beyond Voatz's documentation, that any such blockchain exists.

It is further unclear if Voatz's system is E2E-V. To the authors' knowledge, E2E-V systems in the research literature usually require a voter to visit a polling place and use a paper ballot (e.g. Scantegrity [25] and StarVote [16]), an out-of-band communication before or after the election (see, e.g., code voting [24] and Remotegrity [76]), or a means of performing cryptographic challenges at submission time (see Helios [10]). Assuming that the PDF sent to the user contains no running code, how the system could possibly achieve E2E-V would be difficult to ascertain, and, while Voatz's FAQ appears to tout voter verifiability, it does not explicitly claim to be E2E-V.

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 without E2E-V, this could allow the user to prove the way they voted — violating the requirements of receipt freeness and coercion resistance. If the receipt arrives as an encrypted PDF, it is unclear how Voatz can prove 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 submitted 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? How does one incentivize voters to perform the challenges required for the verification system to be effective? We further note that many of these questions are rooted in open research problems in the E2E-V space [20].

Transparency in design here would help elections officials and 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 U.S. 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 from the vendor, and, while they disputed the severity of the issues, they appeared to confirm the existence of the side channel vulnerability, and the PIN entropy issues.[14] We also spoke directly with affected election officials in an effort to reduce the potential for harming any election processes.

Bug Bounties as a Transparency and Auditing Tool: As previously mentioned, we analyzed the most recent version of the app available in the Google Play store as of January 1, 2020. Voatz also provides a "bug bounty" version of the app via a third party service called HackerOne [5]. The company touts the bug bounty as evidence of Voatz's commitment to independent audits, as well as "community vetting" of the product [6]. We chose not to examine this version of the app for several reasons.

First, evaluating the 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. Worse, all apps are independently randomly obfuscated such that static analysis of each requires a lengthy manual deobfuscation process, so

[14] 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.

repeating this work on a second app represents significant additional effort.

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 the decision to analyze the live app was made, both Voatz's bug bounty app and the Google Play app failed to connect.

Finally, the terms of the bug bounty contain untenable restrictions that hinder an open dialog about the system. For example, the bug bounty excludes both MITM attacks and attacks requiring physical access to the device. This physical access restriction could be read to exclude all of our on-device attacks — To simulate an attacker with access to a remote root-level vulnerability, we used a manual jail-breaking technique which happens to require physical access. The MITM restriction would similarly put the sidechannel attack, as well as the analysis of an adversary that controls Voatz's API server, explicitly out of scope. Worse, the bug bounty, in coordination with their "responsible disclosure policy," also denies researchers safe harbor unless they wait to disclose their findings until some arbitrary time that Voatz decrees the bug fix to be fully deployed [8].

In short, the bug bounty appears to restrict the researcher from disclosure, fails to provide adequate resources for analysis, and arbitrarily considers whole classes of realistic vulnerabilities outside of the scope of the exercise. We conclude that the bug bounty is not particularly relevant for allowing researchers to vet, audit, or improve the system's security, and serves as an example of how such engagements may not be as effective as one may hope. If the goal is to maximize the utility of audits and increase transparency through a bug bounty, vendors could provide source code for both the server and client, publish full system implementation and operational details, and explicitly free researchers to divulge their findings after the industry-standard 90 days, or, at the very least, on a fixed, publicly-available time schedule.

A Note on the Importance of Transparency: The lack of public source and incomplete documentation exacerbate many of the security and information privacy risks documented in this paper, and serve as an example of the importance of transparency in election software. While we had to expend considerable time and effort to deobfuscate Voatz's app and make the results accessible for analysis, the flaws themselves are hardly novel – sidechannel attacks are well known in the cryptographic engineering and research literature, and many of the other issues appear to be the result of poor design and nonstandard implementation. Open access to their code, system design, and running test implementations would have likely revealed these flaws rapidly and encouraged Voatz to fix them, or at least dissuade election officials from putting the voting public at risk.

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. 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 both to avoid disrupting their systems and to comply with the law. Criminals or foreign intelligence agencies, on the other hand, are not constrained to follow U.S. law and would likely have no qualms about disrupting normal operations, including by connecting to Voatz's servers or attacking Voatz directly. Such adversaries will therefore have an *easier* time discovering exploitable vulnerabilities, and are more free to explore flaws we were unable to investigate; 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 explicit disclosure specifying exactly what voter information is collected, how it is used, how long it will be retained, and what third parties may have access constitutes a sharp deviation from privacy best practices, and is an especially concerning omission given the sensitivity of voting information. As mentioned in §4.2, the only notification to the user that Jumio exists is the faint logo placed in the lower right corner of the app's photo screen, and we found no user-accessible indication that Zimperium or Crashlytics are used at all. While the privacy policy does state that Voatz "may transfer Personal Information to third parties for the purpose of providing the Services," it never discloses what information or to whom. Without knowledge of where their personal information is going, there can be no informed consent — as it stands, even the most diligent and privacy-focused individual is likely to misunderstand and assume that their data, particularly their ID information, is only being shared with Voatz.

While Voatz does have a privacy policy, its lack of transparency on important privacy practices such as third-party data sharing leaves voter data unprotected. Beyond serving as a notice to consumers, privacy policies are a critical part of the privacy protection framework, especially in jurisdictions such as the United States that lack comprehensive privacy laws; individual commercial privacy is generally protected in the U.S. only if companies make concrete commitments in their stated privacy policies [67]. For example, because Voatz does not place any explicit data retention time limits on Jumio in a publicly-visible privacy policy, users are at risk of having sensitive election-related information held indefinitely. Barring local statutory restrictions and/or contractual obligations unknown to the authors, the lack of a concrete privacy policy renders Voatz and their partners unaccountable for such privacy failures, and makes it unclear if Voatz can use the information outside of the context of the election itself.

Conclusion: Beginning with West Virginia, Utah, and Colorado, the U.S. 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. [...] I am solving for the problem of turnout. [73]*

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.[15] 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 [29, 41, 60], and of internet voting in general [58], 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 [61] systems using voter-verified paper ballots and Risk Limiting Audits [52] — remain the most secure option. It is the burden of the developer to prove that their system is as secure as these well-vetted methods, to both the public and the security community, before it can be trusted as a crucial component in the democratic process.

[15] Indeed, it is unclear if mobile and internet voting actually increases voter turnout. A study from Switzerland [38] finds, somewhat surprisingly, no statistically significant increase in voter participation.

Postscript

A preprint of this paper was publicly disseminated on February 13th, 2020 and covered in press reports [64]. As a result of our findings, Mason County, Washington, announced it would discontinue using Voatz, followed quickly by West Virginia [26].

Instead of addressing the vulnerabilities reported in this paper, Voatz responded by attacking the credibility of this analysis. In both a public press call [9] and in a blog post entitled "Voatz Response to Researchers' Flawed Report," [72] company officials downplayed the severity of the findings, impugned our intent as well as this paper's overall methodology, and claimed that we examined an outdated version of the app – but oddly never denied the findings themselves.

On March 13, 2020, Trail of Bits, a third-party security firm, released a document detailing a white-box security analysis of Voatz [3]. Their analysis cites this paper, confirms the veracity and severity of all findings reported here, and explicitly contradicts Voatz's criticism — supporting our methodology as an industry-standard process and affirming that there were no security relevant differences between the app we examined and the internal master. Trail of Bits also confirmed that the server-side code contained further vulnerabilities opaque to us (finding 48 issues in total) and that Voatz's protocol is not E2E-V, found no evidence of the mixnet claimed by Voatz, and reported that Zimperium was entirely disabled in at least one of their most recent pilots. Finally, HackerOne has since removed Voatz's bug bounty from their platform – a company first – citing concerns around Voatz's apparent inability to interact in good-faith with security researchers [53].

Despite the findings of the Trail of Bits audit (funded by Voatz) Voatz's CEO continues to publicly deny the veracity of our findings, claiming that "there are like so many errors in the MIT report, that it's just really really hard to accept that report" [36].

Acknowledgments

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, Joseph Kiniry, Barbara Simons, David Jefferson, Neha Narula, Sunoo Park, Ron Rivest, Charles Stewart, and Gerry Sussman for providing feedback and insight.

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. ibotpeaches.github.io/Apktool.

[2] Magisk manager. https://magiskmanager.com/.

[3] Our Full Report on the Voatz Mobile Voting Platform | Trail of Bits Blog. https://blog.trailofbits.com/2020/03/13/our-full-report-on-the-voatz-mobile-voting-platform/.

[4] Realm. https://realm.io/.

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

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

[7] Stay Secure with Jumio's Certified 3D Liveness Detection. https://www.jumio.com/about/press-releases/3d-liveness-detection/, 2018.

[8] Voatz Security Issue Disclosure Policy. https://blog.voatz.com/?p=1278, August 2018. Library Catalog: blog.voatz.com Section: Technology.

[9] Voatz Open Press Call Transcribed from February 13, 2020, February 2020. Library Catalog: blog.voatz.com Section: US.

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

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

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

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

[14] 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.

[15] 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.

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

[17] 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.

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

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

[20] 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.

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

[22] Joseph A. Calandrino, Ariel J. Feldman, J. Alex Halderman, David Wagner, Harlan Yu, and William P. Zeller. Source code review of the Diebold voting system. *University of California, Berkeley under contract to the California Secretary of State*, 2007.

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

[24] David Chaum. Surevote: technical overview. In *Proceedings of the workshop on trustworthy elections (WOTE'01)*, 2001.

[25] 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.

[26] Kevin Collier. West Virginia backtracks on using smartphone voting app in state primary. Library Catalog: www.nbcnews.com.

[27] 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/.

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

[29] 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.

[30] S. Delaune, S. Kremer, and M. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, pages 12 pp.–42, July 2006. ISSN: 2377-5459.

[31] 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.

[32] 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.

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

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

[35] 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.

[36] Lorenzo Emanuel Maiberg Franceschi-Bicchierai, Jason Koebler. A Mobile Voting App That's Already in Use Is Filled With Critical Flaws, March 2020. Library Catalog: www.vice.com.

[37] 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.

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

[39] 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.

[40] 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/.

[41] Rachel Goodman and J. Alex Halderman. 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.

[42] 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.

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

[44] 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.

[45] 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.

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

[47] 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.

[48] 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.

[49] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pages 27–40. IEEE, 2004.

[50] 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.

[51] 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.

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

[53] Sean Lyngaas. HackerOne cuts ties with mobile voting firm Voatz after it clashed with researchers, March 2020.

[54] 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.

[55] 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.

[56] 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.

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

[58] Engineering National Academies of Sciences and Medicine. *Securing the Vote: Protecting American Democracy*. The National Academies Press, Washington, DC, 2018.

[59] 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.

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

[61] 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.

[62] 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.

[63] 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.

[64] Matthew Rosenberg. Voting on your phone: New elections app ignites security debate.

[65] 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.

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

[67] Daniel J Solove and Woodrow Hartzog. The ftc and the new common law of privacy. *Colum. L. Rev.*, 114:583, 2014.

[68] 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.

[69] 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.

[70] 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.

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

[72] Voatz. Voatz Response to Researchers' Flawed Report. https://blog.voatz.com/?p=1209, February 2020.

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

[74] 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.

[75] 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.

[76] Filip Zagórski, Richard T. Carback, David Chaum, Jeremy Clark, Aleksander Essex, and Poorvi L. Vora. Remotegrity: Design and use of an end-to-end verifiable remote voting system. In *International Conference on Applied Cryptography and Network Security*, pages 441–457. Springer, 2013.

[77] 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 Example JSON for a Vote Submission

Figure 10 contains the entirety of the decrypted payload for a vote submission and parameters returned in our synthetic election.


```
{ "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 10: The above is the entirety of the decrypted payload for a vote submission in our synthetic election.

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. 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 Firebase / Crashlytics, gson, protobufs, zxking
- Square OkHTTP & Retrofit
- Datatheorem's TrustKit
- Facebook's SoLoader & Fresco
- Keepsafe's relinker
- Samsung's knox libraries
- Microblink's data capture libraries
- Takisoft's Preference Manager
- 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
- 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