Developing a Personal Voting Machine for the Estonian Internet Voting System

Valeh Farzaliyev[1,2,3], Kristjan Krips[1,3], and Jan Willemson[1,2]

[1]Cybernetica, Narva mnt 20, Tartu, Estonia
[2]STACC, Narva mnt 20, Tartu, Estonia
[3]Tartu University, Inst. of Comp. Sci., Narva mnt 18, Tartu, Estonia

Abstract

As the world's population is increasingly mobile, remote voting becomes more and more a necessity. Vote casting by paper mail is slow, expensive and error-prone. Voting over Internet, on the other hand, is a subject to a wide range of cyber attacks. One of the weakest points in current Internet voting (i-voting) schemes is the end user environment that is hard to control against both vote integrity and privacy attacks. The essence of the problem is that conventional end user devices (PC-s, mobile phones, etc.) are in some sense too powerful, being able to run malware without the voter having efficient means to detect it.

In the current paper, we propose a solution to this problem by introducing a single-purpose user-controlled voting device built on top of a microcontroller platform. We take the Estonian i-voting protocol as the example use case and build an independent client for it that runs on the ESP32 platform. As a by-product, we will also release the first open-source voting client for the Estonian i-voting protocol.

1 Introduction

The primary goal of elections is to adequately reflect political preferences of the electorate. Reaching this goal relies on the election organisers' ability to guarantee integrity of the ballot boxes and transparency of the counting process.

In case of electronic (and especially remote electronic) voting, the nature of such guarantees differs substantially from the paper voting. To start with, there is not even necessarily a physical ballot box that everyone can look at. To compensate for that, a number of verification mechanisms have been proposed [3]. These can be used to verify different claims about the integrity of the vote and tally [34].

Extensive verification can solve the issues regarding integrity and transparency of the election results, but on the other hand, too strong of an evidence (a *receipt*) that the voter's preference was included in the tally can be used to bribe or intimidate the voter. Such coercion issues, in turn, affect voting freedom. Thus, designing a good (remote) voting system requires establishing an acceptable balance between the transparency and vote privacy requirements.

A special class of electronic voting privacy problems stems from malware that can potentially reside on voter's device. On one hand, individual verifiability mechanisms can be used to catch the malware attempts of vote manipulation. At the same time, privacy violations (e.g. sending the voter preference to attacker's server) do not have visible side effects for the voter and can hence remain unnoticed. Thus, the main measure against automated privacy attacks is to restrict the attack surface potentially used by the attacker to successfully deploy malware on voter's device.

In this paper we are going to concentrate on the Estonian i-voting system. Estonia is one of the few countries that provides the option to participate in the national elections by casting the vote over the Internet.

In Estonia, vote casting over Internet has been an option since 2005, and during the 2019 Parliamentary elections, about 44% of all the voters used this medium to cast their vote[1]. Since 2013, the voters have an option of verifying that their vote reached the digital ballot box (vote collection server) as intended [20]. In 2017, a new framework code-named IVXV was deployed making the server-side components considerably more verifiable by external independent auditors [19].

Vast majority of the Estonian i-voting system components were open-sourced in 2013.[2] The only notable exception is the voting client that is used to prepare and cast the electronic vote. The main reason for such a choice is the consideration that the voter's computer is potentially a hostile environment that can be controlled by malware. To make the potential attacker's work at least a bit more complicated, the source code of the client application has never been released. A determined attacker can still manipulate the vote as has been demonstrated a few times (e.g. using a phoney overlay application [18] or a debugger-based approach[3]). However, the fact that the voting client remains closed-source indicates that the election organisers still believe this measure helps to reduce the attack surface. Also, due to the option of individual vote verification, the voter does not need to rely on the voting client solely as a black box.

Still, civil activists, researchers and OSCE/ODIHR missions have criticised such a state of affairs from the viewpoint of transparency [39, 1]. Essentially, we face yet another manifestation of the security-transparency trade-off. Letting a fully open client run on top a potentially untrusted computer would enable a lot of attacks, but at the same time keeping the source closed raises hard-to-answer questions. In our view, this discrepancy occurs due to the general-purpose

[1]`https://www.valimised.ee/en/archive/statistics-about-internet-voting-estonia`
[2]`https://github.com/vvk-ehk/ivxv`
[3]Märt Põder *How I hacked e-elections a little bit*, in Estonian: `https://gafgaf.infoaed.ee/posts/minu-evalimised/`

computers being in some sense too powerful so that malware can operate on them too easily.

In the current paper we propose solving this contradiction by developing a proof-of-concept client that would run on a much less capable computing device. The Estonian voting system was taken as a basis for the proof-of-concept as the back-end source code is public along with the documentation. However, the same idea could be applied for other voting systems where the client device has to be trusted.

By using special purpose hardware it is possible to lower the probability of malware infections as the device is not used for everyday activities and does not run other applications. Our contribution includes the first publicly available voting client for the Estonian i-voting system along with its source code and usage instructions.

On the other hand, our work can be seen as a new development in the area of voting machines. As hand counting paper ballots is a tedious and inherently error-prone activity [21], election organisers have searched for better alternatives. A tempting idea is to get rid of the paper ballots and let a machine record the votes directly. However, the history of Direct Recording Electronic (DRE) equipment is rich in poor design choices and the resulting vulnerabilities [14, 7, 16, 4, 6, 5].

One of the core problems with the DRE machinery is that the voter has little or no control over it. She can not verify that the software running on the machine is genuine, or that the software behaves according to the specification. One way around this problem is having an immutable trail (e.g. a paper receipt with a verification code) that the voter can later check against a bulletin board, as implemented e.g. in Wombat [8], STAR-Vote [9] or vVote [12]. Another approach is to create special hardware for supervised voting systems that is by design secure against known classes of hardware vulnerabilities that could be exploited through software.[4] Wagner *et al.* have researched the options of building DRE machines in an independently verifiable manner [36, 40].

An alternative would be giving the voter greater control over the vote casting device. In 2009, Öksüzoglu and Wallach presented a design based on Xilinx Spartan-3E 500 FPGA platform [32]. Back then, the device was priced around $150, and in order to replicate the build, rather a good knowledge of FPGA frameworks was required. Also, the authors did not make the source code available.

In 2014, Lipmaa described an improvement to Estonian i-voting scheme based on more advanced ID-cards with a small display and integrated PIN-pad [26]. Unfortunately, such ID-cards have not become available in Estonia.

There have been also other proposals to use hardware devices in various functions in the voting process. For example, Islam *et al.* used Raspberry Pi with a camera to visually analyse the voter's ID-card [22], whereas Knutti *et al.* built a classroom quiz and poll system based on TI CC2530 RF chip [24].

In 2015, Grewal *et al.* proposed a system called Du-Vote [17] relying on

[4]`https://freeandfair.us/ssith-secure-hardware-demo/`

voting-specific cryptographic hardware tokens being distributed to all the voters. The authors claimed strong security guarantees, later disputed by Kremer and Rønne [25]. However, the biggest practical problem of Du-Vote is production and distribution of such single-purpose devices. For example it will be rather difficult to ensure that all the tokens will be delivered to the eligible voters only. Comparison with similar devices used for online banking is only partial as voting is rather a rare event. Hence, the cost of production and distribution of such tokens per usage is much higher for voting than it is for banking.

In 2020, we have more options for selecting hardware. For the current paper we took a low-cost general-purpose IoT microcontroller platform ESP32 developed by Espressif[5] and implemented a fully functional voting client for the Estonian i-voting system on top of it. The voter is free to choose her own favourite ESP32 board from a number of suppliers available on the market, and compile the application for it with minimal modifications. When developing our code, we also had ease of portability in mind, hence with some more effort it should be possible to run our voting client on other platforms, too.

We acknowledge that this is not an option for a regular voter. Assembling such a device requires technical skills, so probably not too many people will be using it. However, we expect those people to play an important role in building the public confidence in Estonian Internet voting as it will finally be possible to cast an i-vote using an open source and auditable client software.

The paper is organised as follows. Section 2 reviews the Estonian i-voting scheme and the threats related to the current voting platforms. Section 3 presents our hardware of choice and Section 4 covers implementation details. Finally, Section 5 discusses the security properties of our solution, and Section 6 draws some conclusions.

2 Estonian i-voting scheme

This section gives an overview of the updated version of the Estonian i-voting protocol, which is in use since 2017. The protocol steps are shown in Figure 1, which depicts both the voting and individual verification protocol used for Estonian i-voting [20, 19].

The process starts with the voter authenticating herself to the server using the national ID-card or mobile-ID. The server checks voter eligibility, finds the corresponding precinct and replies with the list of candidates L of that precinct. The PC based voting software displays the names of the candidates. The voter makes her choice c_v, voting software encrypts it with the election's public ElGamal key (using encryption randomness r, which the software generated locally). Next, the voter confirms the choice by signing the encrypted ballot with the Estonian ID-card or mobile-ID. The encrypted and signed ballot $Sig_v(Enc_{s_{pub}}(c_v, r))$ is sent to the vote storage server.

In order to facilitate individual verification, the server replies with a vote reference vr which the voter can transfer to her smartphone together with the

[5] `https://www.espressif.com/en/products/hardware/socs`

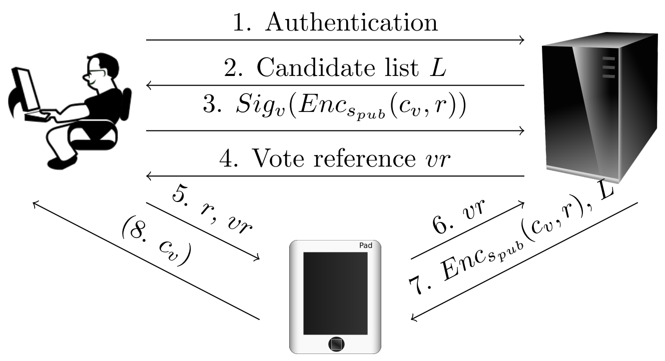


Figure 1: Estonian i-voting and verification protocol [20].

encryption randomness r by using a QR code (displayed in Figure 2). The voter has a limited time window (30 or 60 minutes depending on elections) to use a smartphone based vote verification application before the vote reference expires. The limitation is added to prevent the verification application from being used for coercion.

After scanning the QR code the verification application on the smartphone queries the vote storage server for the vote cryptogram, verifies its signature and time-stamp, and decodes the vote using r (which is straightforward due to the properties of ElGamal encryption)[6]. The value of the vote found from the cryptogram is finally displayed to the voter on the mobile device screen so that the voter can take a decision whether this value matches her original intent.

I-voting is only available during the advance voting period. The voter can change her i-vote as many times as she wants in case she feels that her previous vote(s) was (were) given under coercion. Only the last i-vote is taken into account in the tally. However, in case the voter casts a paper vote the i-votes are cancelled.[7]

2.1 Threats related to the voting device

If the voting protocol is not specifically designed to protect privacy of the vote during the casting process, it is very hard for the voting client software to hide the vote from a malicious voting device. Malware can capture either a screenshot or a video[8] and thereby record the choice of the voter. Such information gathering abuses core system functionalities and does not require elevated permissions.

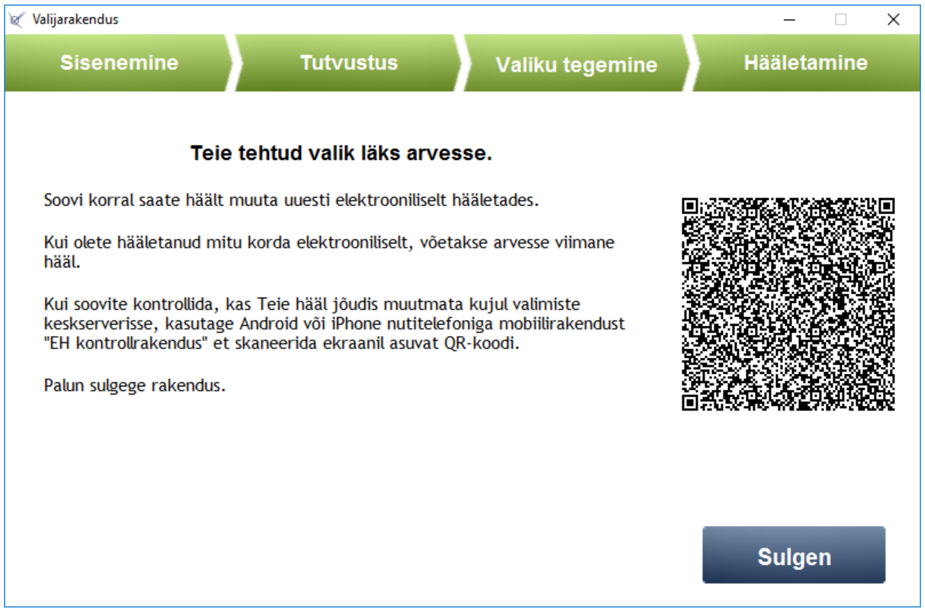
[6]`https://github.com/vvk-ehk/ivxv/blob/master/Documentation/en/protocols/05-ehaal.rst\#encrypted-ballot`

[7]Up to 2019, changing one's i-vote by voting on paper was only an option in the advance voting period. Starting from 2021, the i-vote can also be changed by paper vote on the election day. `https://www.riigikogu.ee/en/sitting-reviews/riigikogu-passed-14-acts/`

[8]`https://attack.mitre.org/techniques/T1125/`

Protecting vote privacy from a malicious voting device requires the identities of the candidates to be hidden from the voting client. This is very difficult to implement with most of the world's voting schemes, including the Estonian voting protocol (unless one is willing to use a hard-to-deploy schemes based on e.g. visual cryptography [28, 35]). As an alternative, code voting could be used, which would replace candidate names with unique codes that are generated for each voter [11]. However, the codes would have to be pre-distributed to the voters via a reliable private channel, which is a non-trivial task in the remote voting setting where the physical location of the voter should not affect the voting process.

There are multiple approaches that allow to protect integrity of the vote. One common solution is to allow the voters to check that the vote correctly reached the vote collection server. Another option is to allow the voter to check that the vote was counted in the tally. The Estonian voting protocol uses the former and allows the voters to verify that the cast vote was not modified and reached the voting system. In the end of the voting process, a QR code is displayed in the voting application as shown in Figure 2.



Teie tehtud valik läks arvesse.

Soovi korral saate häält muuta uuesti elektrooniliselt hääletades.

Kui olete hääletanud mitu korda elektrooniliselt, võetakse arvesse viimane hääl.

Kui soovite kontrollida, kas Teie hääl jõudis muutmata kujul valimiste keskserverisse, kasutage Android või iPhone nutitelefoniga mobiilirakendust "EH kontrollrakendus" et skaneerida ekraanil asuvat QR-koodi.

Palun sulgege rakendus.

Figure 2: The view of the official voting application after a vote has been cast.[10]

The verification information (vr, r) that is contained in the QR code can be

[10]`https://www.valimised.ee/en/internet-voting/checking-i-vote` The text says that it is possible to change the vote by re-voting, but only the last vote will be taken into account. It also says that one can use the verification application "EH kontrollrakendus" on an Android device or on an iPhone to check if the vote correctly reached the voting server.

read and queried by an open source verification application[11], which is available for both Android and iOS devices.

However, verification offers a reactive measure that only helps to detect some classes of attacks. The Estonian vote verification protocol allows the voter to verify the vote up to three times during a limited time window, but in case a re-vote is cast, the verification application still successfully verifies the receipts of the previous votes which are already overruled. At first, this behaviour may seem to be counter-intuitive, but the reasoning becomes apparent when coercion resistance is considered. The re-voting functionality is designed to counter coercion and thus the vote verification functionality can not be used by the coercer to check if the coerced vote was overwritten. As a negative side-effect, the verification application is not always able to detect if the vote was overwritten by malware.

The verification system only allows to detect malware that either modifies the vote before it is being submitted or prevents voter's ballot from reaching the voting system. This is still a significant functionality as an attacker is not only motivated to modify the votes, but also to avoid being detected. From the attacker's perspective, it is safer to either locally drop or modify the vote instead of casting a re-vote as server side re-voting logs can be analysed to detect anomalies. Therefore, the attacker has motivation to evade verification. However, even if the voter verified the vote and got an unexpected result, additional user interaction is required to report an error. More specifically, in case an anomaly is detected during vote verification, the voter is instructed to contact the election organizers by either sending them an email or calling them to report the error.

The rationale behind the architecture of the Estonian vote verification system is that if a large enough random subset of the voters verifies their votes, a large scale attack can be detected and the i-voting either postponed or cancelled. By following that logic, the attacker should have less motivation to initiate the attack.

Throughout the years, since introduction in 2013, vote verification has been used on average by about 4% of Estonian i-voters. However, it has also been shown by Solvak that socio-demographic and behavioural parameters of voters can be used to profile the potential verifiers, at least with some better than random guessing probability [37]. A good profile of the potential verifiers can help the attacker to select the potential victims of vote manipulation attack, lowering the risk of being detected.

If profiling is still not precise enough, an attacker could use simpler means to evade verification. One simple approach for the attacker would be to interfere in the voting process and crash the voting application just after the first vote is cast to prevent verification, hoping that the voter does not bother to re-vote. Another option for the attacker would be to measure the time the voting application is running after the vote has been cast. In case the voting application is closed right away, it may be likely that the voter did not scan the QR code

[11]`https://github.com/vvk-ehk`

and the attacker could use that information to prevent the vote from being sent to the voting system. More specifically, the attacker may interfere with the process by delaying the vote from being sent to the voting system until it can be guessed whether the voter is going to verify the vote.

An attacker could also use malware to access the electronic identity of the voter in order to maliciously cast a vote. Voters are not given an option to check if their vote was included in the tally, nor are they notified of a re-vote being cast as these functionalities would open up opportunities for e.g. vote selling. Therefore, the voter does not have any means to check whether the malicious voting device has cast a re-vote on her behalf.

It is unclear how many votes would have to be modified or dropped to change the election outcome in Estonia as this has not been studied. Recent studies in Australia [10] and Belgium[12] showed that changing only a small amount of votes can significantly influence the election outcome. The results of such studies depend on the used election system and on the local political landscape, but they are easy to conduct as the data is publicly available. Thus, for a future study it would be interesting to find out how many votes would have to be modified to change the number of ways to form a coalition. Such information could be used as a basis for further studies to find the balance between coercion resistance properties and integrity protection mechanisms.

The previously mentioned attacks are just some of the threats that have to be considered when voting from an untrusted device. There are two main approaches for mitigation. One would be to redesign or modify the voting protocol. Another is to lock up and control the environment where the voting client is used. The latter is currently not feasible due the design of the modern operating systems, although iOS tends to move towards that direction. Thus, we decided to take a shortcut to the second approach by creating a proof-of-concept solutions that uses a microcontroller as the vote casting platform. While a microcontroller offers a limited environment, there are still components and software which have to be trusted or verified. The trust base of the voting device is discussed in Section 5.

3 Design and choice of hardware

This section describes how the design choices were influenced by the software and hardware level requirements.

As mentioned in Section 1, the source code of the Estonian i-voting client application is not publicly available. However, the protocol description on the API level is public, together with the corresponding server-side implementation.[13]

The voting client and the back-end communicate over TLS using JSON-RPC. Such communication is required when:

- authenticating the voter,

[12]`https://decryptage.be/2019/10/deux-voix-pour-changer-une-majorite/`
[13]`https://github.com/vvk-ehk/ivxv`

- fetching the list of choices,
- fetching the signing certificate (JSON-RPC is used with mobile-ID; certificate for ID-card is available locally),
- signing the encrypted ballot (JSON-RPC is used with mobile-ID; with ID-card, signing is a local operation),
- sending the encrypted and signed ballot to the vote collection server,
- getting the OCSP response along with the vote reference, which is required for verification.

In addition, the voting client has to be able to export the encryption randomness and the vote reference in order to allow the ballot to be verified. Thus, the voting client should either display these values as a QR code like it is done in the official voting client, or export these values in some other form that the verification application can process.

The main goal of our project was to provide a proof-of concept open-source voting client working on a minimal set of hardware assumptions. Still, the hardware must be powerful enough to be able to perform the necessary cryptographic operations, most notably 3-kilobit ElGamal encryption used to encrypt the votes in the current IVXV implementation.

The hardware requirements were fulfilled by ESP32 module, which is a well-known microcontroller created by Espressif Systems. ESP32 has a dual-core 32-bit Xtensa® LX6 microprocessor with clock speed up to 240MHz. Other features of the module include: Wi-Fi (2.4 GHz band), Bluetooth 4.2, 40/80 MHz flash frequency, 4 MB flash memory, several peripherals (I2C, SPI, UART, etc.) and in some versions a 128×64 pixel 0.96 inch integrated OLED screen.

The user has to interact with the ESP32 to select the candidate. In the simplest set-up, we decided to use a KY-040 rotary encoder that allows to both scroll and click (see Figure 5). These functions are sufficient to select a candidate from the list that is displayed on the 0.96 inch integrated OLED display. However, this screen is not sufficiently large to display verification QR code.

Alternatively, a touchscreen can be used to provide input and a sufficiently large area to display the QR code. This approach is described in Section 4.2.

In case a large screen is not available, the voting device must export the information required for verification using some other medium. It is important to protect the verification information as it can be used to reveal how the voter voted. Thus, exporting this information to a general purpose computer that may contain malware is not a good option.

We developed an alternative verification mechanism for the configuration where touchscreen is not available. We took the source code of the official vote verification application as a basis and modified it so that it can read the vote reference and randomness over Bluetooth. This approach is described in

Section 4.1. We published the source code of the modified vote verification application so that its behaviour can be validated by the community.[14]

4 Implementing the personal voting device

The ESP32 based voting client is written in C++ using Eclipse IDE with the open source ESP-IDF framework[15] provided by Espressif.

There are multiple different microcontroller boards that integrate ESP32 module. We tested the voting application on the following boards: DOIT DevKit v1, Wemos Lolin32, Heltek ESP32 Wifi Kit. It is important to note that the pin configuration and hardware set-up may differ depending on the selected board. Detailed information about the configuration of the tested boards is presented in the Appendix A.

Our implementation is modular and allows to easily replace the components (e.g. rotary encoder or touchscreen) when the corresponding drivers are available. Some modifications may, of course, be required. In fact, we had to tweak the KY-040 rotary encoder module's unofficial driver library for ESP-IDF framework a little to support the click functionality.

As mentioned above, we created two example set-ups. The first, subsequently called *basic*, is composed of a board with 128×64 integrated monochrome OLED screen and KY-040 rotary encoder. The other set-up (called *extended*) uses an external 240×320 colour TFT LCD screen with a touchscreen breakout board.

In both of the cases, we have only implemented support for mobile-ID authentication and signing. This is sufficient for i-voting and does not require integrating extra hardware, as opposed to the need for a card reader in case of ID-card.

4.1 Basic set-up

For basic set-up, we used boards with built-in displays, i.e. Wemos Lolin32 OLED and Heltec ESP32 Wifi Kit. Their main difference is the pin layout and crystal oscillator frequency, see Table 3 in Appendix A. Figure 3 presents the schematics of the layout.

The resolution of the 0.96 inch integrated screen is not sufficient to show a QR code, instead we used Bluetooth to transmit the data required for verification. As the original verification application[16] does not support Bluetooth connections, we forked and modified it. The Bluetooth channel has to be secured to keep the voter's choice private. Thus, instead of relying on the traditional numerical comparison method for Bluetooth pairing, users are asked to manually enter the confirmation PIN displayed on the screen of ESP32. After successful

[14]https://github.com/Valeh2012/PersonalVotingMachine
[15]https://docs.espressif.com/projects/esp-idf/en/latest/esp32/
[16]https://github.com/vvk-ehk/ivotingverification

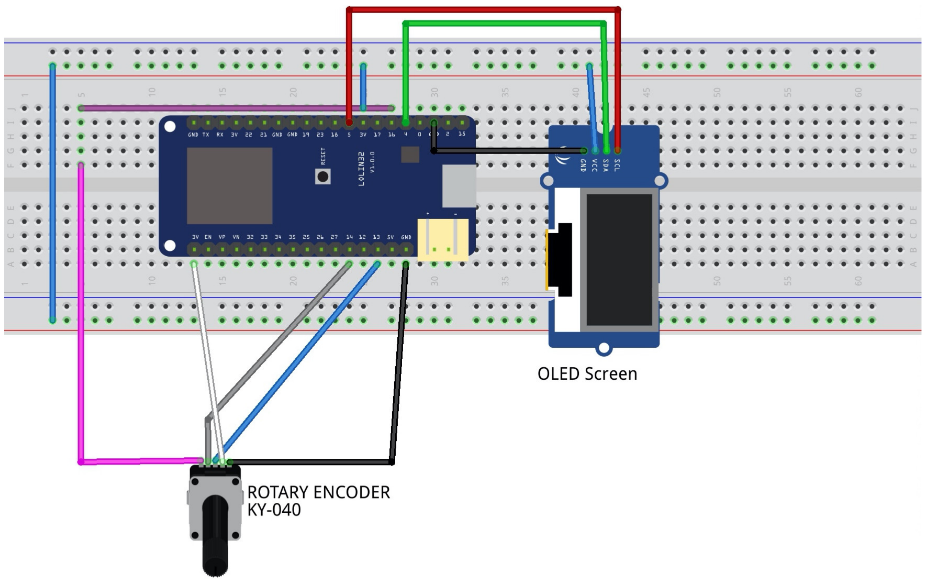


Figure 3: Schematics of the basic set-up.

data transmission, the connection is closed and the connected device is removed from the bond list.

Although, the basic set-up provides the necessary functionalities, it adds additional components that have to be trusted, which may not be an optimal solution. For example, Bluetooth is famous for its lengthy specification[17] and list of vulnerabilities[18]. However, even when an attacker would find a vulnerability that would allow to intercept Bluetooth communication, the integrity of the vote would not be affected.

There are two approaches for removing the Bluetooth dependency. First, in the basic set-up, one could replace Bluetooth channel with NFC channel. However, that would require additional hardware as ESP32 does not have a built in NFC reader. We did not implement this, as the official verification application does not support NFC and thus the voter would still have to rely on a modified verification application. A simpler solution is to remove the Bluetooth dependency by using the extended set-up, which is described in the next section.

4.2 Extended set-up

The main problem with the basic set-up is that the screen size is not large enough to accommodate a QR code for vote verification. Thus, the voter has to install a modified version of the verification application. This approach is not

[17] `https://vtsociety.org/wp-content/uploads/2019/07/Core_v5.1.pdf`
[18] `https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Bluetooth`

only complicated, but it also increases the trust base as the voter has to either trust the modified verification application, or verify its source code and build the application on her own. Therefore, we also describe an alternative set-up that is compatible with the official vote verification application.

The extended set-up uses ILI9341 as touchscreen. Its 240 × 320 pixel screen fits the QR code and thus supports the official vote verification application. Thereby, the Bluetooth dependency is removed from the extended set-up. The touch support also allows to remove the rotary encoder and perform all the required user interaction on the screen.

Figure 4 shows the connection schematics. The pin layout for the extended set-up is described in the Appendix A.2.

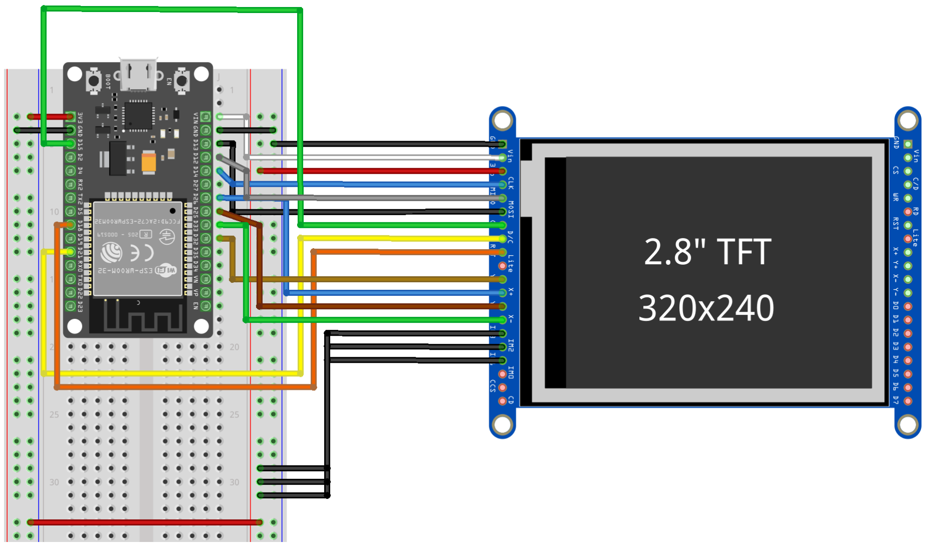


Figure 4: Extended set-up schematics.

4.3 Implementation challenges and performance

Implementing the voting device software was rather challenging. Although API documentation for IVXV was available and easy to understand, it was not always sufficiently detailed. We found and reported several minor mistakes in the IVXV API documentation. However, that was not the only challenge. When writing the implementation the device limitations also needed to be taken into account.

Our ESP32 modules came with 4MB flash memory. If an extra SD card is not used, the 4MB should host the application, operating system and RAM. By default, ESP-IDF creates a factory partition of size 1MB to store application binary. As our client application of 1.3MB did not fit into this partition, its size had to be changed to 1.5MB.

ESP32 contains DRAM (Data RAM), IRAM (Instruction RAM) and D/IRAM (can be used as DRAM or IRAM). RAM should be used carefully to fit both application data and executable data (instructions). Application data stored in DRAM includes zero-initialized and non-constant static data, stack and heap buffers. Enabling Bluetooth reduces DRAM size by 64KB. On the other hand, timing critical code and interruption handlers should be placed into IRAM, which also hosts CPU cache, FreeRTOS library, some components of ESP-IDF and WiFi stack. During application boot-up, internal components mentioned above are mapped into the memory. The remaining memory buffers can be controlled by the application. We measured this portion of memory to be less than 200 KBs for both set-ups. As a result of optimisations, the application uses less than 60 KB of heap in total (<20KB of heap is used to store digital signature files in zip format, the rest is used during computations). Table 1 summarizes the heap usage in case of different set-ups.

Table 1: In the basic set-up, the slash separated values denote memory usage in Wemos Lolin32 OLED and Heltec ESP32 WiFi Kit boards, respectively. The referred components include WiFi, rotary encoder and/or (touch)screen.

Feature	Heap usage (Bytes)	
	Basic set-up	Extended set-up
Components	37944 / 37996	36352
Encryption	2644 / 2640	2448
Signature	11748 / 11596	12584
Total	47756 / 45600	59900

To benchmark the implementation it is sufficient to measure the performance of ElGamal encryption, which is used to encrypt the ballot. Authentication and signing the hash of the encrypted ballot are offloaded to voter's smartphone as they are done with voter's mobile-ID. The rest of the voting client's functionalities take a non-significant amount of computing time.

IVXV uses 3072-bit ElGamal to encrypt the votes. Our implementation uses the modular exponentiation primitive from mbed TLS library, which is supplied as a part of ESP-IDF.

By pinning tasks to a specific CPU core in FreeRTOS it is possible to parallelize computations. We used both cores to compute the ElGamal cryptogram

$$E(pk, m) = (c_1, c_2) = (g^r \bmod p, m \cdot pk^r \bmod p),$$

to obtain maximal speed-up as each component of the pair (c_1, c_2) is calculated independently. This resulted in the usage of additional 1.3KB of heap and 2KB of stack memory. The encryption randomness r is generated using the `esp_random()` function that outputs strong random values if either WiFi or Bluetooth is enabled[19]. As our application requires WiFi to work anyway, this assumption is satisfied.

[19]https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/system.html

Additionally, it is possible to turn on hardware acceleration for cryptographic operations on ESP32. Table 2 compares the time ElGamal encryption takes for different configurations and optimization methods. Since the CPU-s of all three of our test boards were the same, the performance results are the same as well.

Table 2: ElGamal encryption performance for different configurations.

	CPU clock speed	Encryption time (s)
Single core	80 / 160 / 240 MHz	40 / 19 / 13
Two cores	160 / 240 MHz	10 / 6
Two cores, hardware acceleration	160 / 240 MHz	1 / 0.66

Testing was conducted against a test server, which was running the same code as the server used during real elections. We were not able to test our application during elections as the next elections in Estonia, which use the i-voting channel are planned for the year 2021.

5 Discussion

The cost of the hardware for the voting machine with the basic set-up is between 15-20 USD depending on where the parts are bought from. This contains the ESP32 with built in OLED, rotary encoder, breadboard and jumper wires. The cost of the extended set-up is slightly higher due to the 2.8 inch Adafruit touchscreen, which adds about 30 USD to the price. As optimizing the price was not one of our goals, we did not try to find cheaper alternatives for the touchscreen.

The low cost and accessibility of the hardware plus the simplicity of assembly may attract third parties to the idea of selling the device. However, we have to emphasize that the goal of this project was to create only a proof of concept. Therefore, we did not consider the risks that could be caused by the distribution, support and supply chain of the device. Although it is possible to use secure boot on ESP32 to protect the integrity of the firmware, it has recently been revealed that this measure can be bypassed if an attacker has physical access to the device.[20] Thus, even though secure boot functionality could in theory be used to secure the supply chain, it may be insufficient in practice.

One result of our research is the publication of the source code for the voting client. That may have implications on the overall security of the Estonian voting system. Third parties could use the source code to build a voting application for insecure devices. This is possible as the voting client is independent from the back-end system and the client software only has to do proper API calls. The distribution of the voting client e.g. for mobile devices can create additional

[20] `https://www.espressif.com/en/news/Security_Advisory_Concerning_Fault_Injection_and_eFuse_Protections`

risks, but these risks can not be avoided even if the source code of the voting client would not be published. The document containing the description of the API for the voting client is public along with the source code of the server side software, which also reveals the API.

It can be argued that the attacker's task may be simplified due to the access to the source code of the voting application. However, the attackers who have the motivation and resources to influence an election already have sufficient information to build an independent voting client. In addition, such threat actors have access to reverse engineering capabilities, which can be used to inspect how the official voting client behaves. For example, a paper published in 2020 by MIT researchers described how they were able to reverse engineer Voatz i-voting application by using Ghidra, regardless of the multiple obfuscation techniques used by Voatz [38].

Although it is more difficult to attack the ESP32 based voting client due to its small trust base, the threat of abusing the voter's electronic identity on other platforms remains as it still depends on the behaviour of the voter. Regardless, the ESP32 based voting client provides a level of ballot privacy that is very difficult to achieve with a regular computer that may not be solely controlled by the voter.

5.1 Trust base

Currently, the Estonian i-voting client is only available for the following three major platforms: Windows, MacOS, Ubuntu (other Linux distributions are not guaranteed to be supported, but some of them work). Thus, these operating systems and the third party software that runs with administrative privileges have to be trusted to not act maliciously. For example, the voter has to trust that the antivirus software is not abusing its access as it may have happened with Kaspersky [33]. As it is common to find vulnerabilities from third party software and the libraries it depends on [13, 2] one also has to hope that the software vendors issue timely patches to prevent the vulnerable software from being used as a launchpad to infect the device. Thus, the end user has to trust the software vendors. However, even if the vendor is proactive and honest, an attacker could target the supply chain by infecting the vendor and thereby issuing a malicious update as it happened with CCleaner [30]. The CCleaner case is remarkable as the malicious update was downloaded over two million times and it remained undetected for a month.

Besides the software that runs with elevated privileges, it is also necessary to trust the less privileged software to not take screen captures of the voting process. In addition, the origin and the integrity of drivers have to be trusted. The trust and integrity problem is usually solved by a signature issued by a trusted certification authority. The trust model in PKI is based on the assumption that the private keys of the certification authorities do not leak and are not abused. Still, there are examples of malicious drivers that have been signed with keys corresponding to valid certificates as it happened with Stuxnet [31]. Alternatively, malicious software can use various tricks to make the verification systems believe that the software was correctly signed [23].

End user's software can use PKI due to a local list of root certificates. In case that list is modified, the whole chain of trust can be circumvented. But modifying the list of trusted root certificates is a common behaviour both in corporate networks and by antivirus software to enable scanning of TLS traffic [15].[21] However, such functionality also allows to break integrity of the channels and for example choose which executable files are delivered to the client devices. Also, by modifying the local list of trusted certificate authorities, a third party is able to generate trust by signing executable files that will successfully verify on the client device without generating errors. All of the aforementioned methods could be used to affect the voting process but the regular voter does not have the knowledge to check the configuration of the computer that is used for voting.

By replacing the modern desktop environment with ESP32, the trust base of the voting device can be significantly reduced. ESP32 allows to run one application at a time, which removes the need to trust third party software. Thus, after flashing the firmware containing the specific application, ESP32 can be considered a single purpose device. As the device is not used for other activities, the contact with the outside world is limited only to the corresponding application and the interfaces which are used to connect with the voter and the voting servers. Therefore, the attack surface for getting remote access to the device is severely limited.

The other attack vectors come either from getting physical access to the device, or from the supply chain, attempting to infect the device and/or its software during production. Out of these, we consider the threat of a physical attack to be very limited, since in our model the voter builds the device herself, flashes it and uses it only for a very limited time in the environment of her own choosing. This also means that even if vulnerabilities are found in the source code, exploiting them is non-trivial due to the limited access.

We estimate that the threat of planting hardware level malware during microcontroller production is low. Still, if the voter considers this as a significant risk, she can port our application onto a platform that she trusts. However, the threat must be re-evaluated in case one would like to scale up the production of such voting devices.

Final part of the trust base consists of the software that is used to build the voting client. The main component of the trusted software is the voting application itself. We reduce its trust base by open sourcing the code so that its behaviour can be verified. In addition, the firmware of the voting application contains one trusted root certificate to support the TLS connection to the voting server. Thereby, we do not have to trust the global PKI. The rest of the trust base comprises of open source third party software which is used to develop and run the voting application. The largest trusted component is the Espressif IoT Development Framework (ESP-IDF) provided by Espressif Systems, used to create applications for ESP32. ESP-IDF is open-source and available from a

[21] `https://blog.cloudflare.com/monsters-in-the-middleboxes/`

GitHub repository.[22] The framework is built on top of FreeRTOS and contains a list of libraries and modules that can be selectively enabled in the firmware of the application being developed (e.g., Bluetooth can be switched off by not including the corresponding libraries during the build).

In order to compile and build an application with ESP-IDF, a toolchain is needed. The toolchain slightly differs depending on the operating system, but the instructions are available from the website of ESP-IDF.[23]

The voting application also depends on the `miniz` data compression library[24] and on ESP32 Arduino core[25]. Other dependencies vary based on the selected approach. In case of the basic set-up, a library for the rotary encoder[26] is needed together with the U8g2 library[27] for the display and UI. However, in case of the extended set-up, the following dependencies are added: QRCode[28], Adafruit Touchscreen library, LittlevGL for display connection and UI.

6 Conclusions and future work

One of the main issues that the remote voting systems face is the untrustworthiness of the voter's computer. This can be partially solved by investing into improving the verification properties of the voting protocol. However, by increasing verifiability and transparency of the system, coercion resistance may be reduced. Therefore, a good balance between these requirements needs to be found that is both applicable in practice and usable by voters.

One way to test this balance is to run usability studies of the verification and anti-coercion mechanisms. Examples of such studies can be found in [27, 29]. Ideally, performing usability studies should be a precondition for implementing a change that affects the aforementioned mechanisms. Eventually, a political decision is needed to set the balance between integrity and privacy guarantees. However, the decision should be based on informed risk analysis of possible options.

The aim of this paper was to build a proof-of-concept instance of what could be called a personal single-purpose voting machine. We used the Estonian i-voting system as an established framework and ESP32 as an example platform for our implementation. Making all the instructions and the source code available enables the voters to build their own voting devices. As a by-product, we built an open voting client for the Estonian i-voting protocol, finally making the Estonian system fully open source.

In addition to improved transparency, the voting device has a significantly reduced trust base and a considerably smaller attack surface. Thus, software

[22] `https://github.com/espressif/esp-idf`

[23] `https://docs.espressif.com/projects/esp-idf/en/latest/get-started/toolchain-setup-scratch.html`

[24] `https://github.com/richgel999/miniz`

[25] `https://github.com/espressif/arduino-esp32`

[26] `https://github.com/DavidAntliff/esp32-rotary-encoder`

[27] `https://github.com/olikraus/u8g2`

[28] `https://github.com/ricmoo/QRCode`

and hardware level vulnerabilities are less significant as they are difficult to exploit.

Of course, our implementation is only an academic proof of concept, not being ready to be used by every voter. However, even a small fraction of voters using such devices would illustrate the transparency of the voting system and show one possible path for combating malware in the context of i-voting.

The ease and low price tag of creating a proof of concept device shows that, in principle, it may be possible to build and distribute cheap personal voting devices in the future. For both ethical and financial reasons the researchers can only make the first step towards building an individual voting device. The initiative for building a voting device that could be securely distributed to the voters has to come from the state. Whether it would be possible to do that securely and in an economically viable matter is an open question for future research and development.

References

[1] Estonia, Parliamentary Election, 6 March 2011: OSCE/ODIHR Election Assessment Mission Report. OSCE Office for Democratic Institutions and Human Rights, 2011.

[2] 2019 state of the software supply chain, 2019.

[3] Syed Taha Ali and Judy Murray. An overview of end-to-end verifiable voting systems. In *Real-world electronic voting: Design, analysis and deployment*, pages 171–218. CRC Press, 2016.

[4] Andrew W Appel, Maia Ginsburg, Harri Hursti, Brian W Kernighan, Christopher D Richards, and Gang Tan. Insecurities and inaccuracies of the Sequoia AVC Advantage 9.00 H DRE voting machine, 2008.

[5] Andrew W. Appel, Maia Ginsburg, Harri Hursti, Brian W. Kernighan, Christopher D. Richards, Gang Tan, and Penny Venetis. The New Jersey Voting-machine Lawsuit and the AVC Advantage DRE Voting Machine. In *2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '09*. USENIX Association, 2009.

[6] Adam J. Aviv, Pavol Cerný, Sandy Clark, Eric Cronin, Gaurav Shah, Micah Sherr, and Matt Blaze. Security Evaluation of ES&S Voting Machines and Election Management System. In *2008 USENIX/ACCURATE Electronic Voting Workshop, EVT 2008*, 2008.

[7] J. Bannet, D. W. Price, A. Rudys, J. Singer, and D. S. Wallach. Hack-a-vote: Security issues with electronic voting systems. *IEEE Security & Privacy*, 2(1):32–37, 2004.

[8] Jonathan Ben-Nun, Niko Fahri, Morgan Llewellyn, Ben Riva, Alon Rosen, Amnon Ta-Shma, and Douglas Wikström. A New Implementation of a Dual

(Paper and Cryptographic) Voting System. In *5th International Conference on Electronic Voting 2012, (EVOTE 2012)*, volume P-205 of *LNI*, pages 315–329. GI, 2012.

[9] Josh Benaloh, Michael D. Byrne, Bryce Eakin, Philip T. Kortum, Neal McBurnett, Olivier Pereira, Philip B. Stark, Dan S. Wallach, Gail Fisher, Julian Montoya, Michelle Parker, and Michael Winn. STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. In *2013 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '13*. USENIX Association, 2013.

[10] Michelle Blom, Peter J. Stuckey, and Vanessa J. Teague. Computing the margin of victory in preferential parliamentary elections. In *Electronic Voting - Third International Joint Conference, E-Vote-ID 2018*, volume 11143 of *LNCS*, pages 1–16. Springer, 2018.

[11] David Chaum. Surevote: Technical overview. In *Proceedings of the Workshop on Trustworthy Elections (WOTE'01)*, 2001.

[12] Chris Culnane, Peter Y. A. Ryan, Steve A. Schneider, and Vanessa Teague. vVote: A Verifiable Voting System. *ACM Trans. Inf. Syst. Secur.*, 18(1):3:1–3:30, 2015.

[13] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on android. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2187–2200. ACM, 2017.

[14] David L. Dill, Bruce Schneier, and Barbara Simons. Voting and technology: who gets to count your vote? *Commun. ACM*, 46(8):29–31, 2003.

[15] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J. Alex Halderman, and Vern Paxson. The Security Impact of HTTPS Interception. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017*. The Internet Society, 2017.

[16] Ariel J. Feldman, J. Alex Halderman, and Edward W. Felten. Security Analysis of the Diebold AccuVote-TS Voting Machine. In *2007 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'07*, 2007.

[17] Gurchetan S. Grewal, Mark Dermot Ryan, Liqun Chen, and Michael R. Clarkson. Du-Vote: Remote Electronic Voting with Untrusted Computers. In Cédric Fournet, Michael W. Hicks, and Luca Viganò, editors, *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 155–169. IEEE Computer Society, 2015.

[18] Sven Heiberg, Peeter Laud, and Jan Willemson. The Application of I-Voting for Estonian Parliamentary Elections of 2011. In *E-Voting and Identity - Third International Conference, VoteID 2011*, volume 7187 of *LNCS*, pages 208–223. Springer, 2011.

[19] Sven Heiberg, Tarvi Martens, Priit Vinkel, and Jan Willemson. Improving the verifiability of the estonian internet voting scheme. In *Electronic Voting - First International Joint Conference, E-Vote-ID 2016*, volume 10141 of *LNCS*, pages 92–107. Springer, 2016.

[20] Sven Heiberg and Jan Willemson. Verifiable internet voting in Estonia. In *6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014*, pages 1–8. IEEE, 2014.

[21] Paul S Herrnson, Michael J Hanmer, and Richard G Niemi. The impact of ballot type on voter errors. *American Journal of Political Science*, 56(3):716–730, 2012.

[22] Md Maminul Islam, Md Sharif Uddin Azad, Md Asfaqul Alam, and Nazmul Hassan. Raspberry Pi and image processing based electronic voting machine (EVM). *International Journal of Scientific & Engineering Research*, 5:1506–1510, 2014.

[23] Doowon Kim, Bum Jun Kwon, and Tudor Dumitraş. Certified Malware: Measuring Breaches of Trust in the Windows Code-Signing PKI. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1435–1448. ACM, 2017.

[24] Fabian Knutti, Nicolas Tobler, and Heinz Mathis. Low-power voting device for use in education and polls employing TI's CC2530 RF CHIP. In *2014 6th European Embedded Design in Education and Research Conference (EDERC)*, pages 221–224. IEEE, 2014.

[25] Steve Kremer and Peter B. Rønne. To Du or Not to Du: A Security Analysis of Du-Vote. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 473–486. IEEE, 2016.

[26] Helger Lipmaa. A Simple Cast-as-Intended E-Voting Protocol by Using Secure Smart Cards. Cryptology ePrint Archive, Report 2014/348, 2014. `https://eprint.iacr.org/2014/348`.

[27] Karola Marky, Oksana Kulyk, Karen Renaud, and Melanie Volkamer. What did I really vote for? In Regan L. Mandryk, Mark Hancock, Mark Perry, and Anna L. Cox, editors, *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*, page 176. ACM, 2018.

[28] Moni Naor and Adi Shamir. Visual cryptography. In *Advances in Cryptology — EUROCRYPT'94*, pages 1–12, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[29] André Silva Neto, Matheus Leite, Roberto Araújo, Marcelle Pereira Mota, Nelson Cruz Sampaio Neto, and Jacques Traoré. Usability Considerations For Coercion-Resistant Election Systems. In *Proceedings of the 17th Brazilian Symposium on Human Factors in Computing Systems*, IHC 2018, pages 40:1–40:10, 2018.

[30] Lily Hay Newman. Inside the Unnerving Supply Chain Attack That Corrupted CCleaner. *Wired*, April 2018.

[31] Eric Chien Nicolas Falliere, Liam O Murchu. W32.Stuxnet Dossier. Symantec Whitepaper, February 2011.

[32] Ersin Öksüzoglu and Dan S. Wallach. VoteBox Nano: A Smaller, Stronger FPGA-based Voting Machine. In *2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '09*. USENIX, 2009.

[33] Nicole Perlroth and Scott Shane. How Israel Caught Russian Hackers Scouring the World for U.S. Secrets. *The New York Times*, October 2017.

[34] Stefan Popoveniuc, John Kelsey, Andrew Regenscheid, and Poorvi L. Vora. Performance Requirements for End-to-End Verifiable Elections. In *2010 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '10*. USENIX, 2010.

[35] A. B. Rajendra and H. S. Sheshadri. Visual Cryptography in Internet Voting System. In *Third International Conference on Innovative Computing Technology (INTECH 2013)*, pages 60–64, 2013.

[36] Naveen Sastry, Tadayoshi Kohno, and David Wagner. Designing Voting Machines for Verification. In *Proceedings of the 15th USENIX Security Symposium*, 2006.

[37] Mihkel Solvak. Does Vote Verification Work: Usage and Impact of Confidence Building Technology in Internet Voting. In *Proceedings of E-Vote-ID 2020*, volume 12455 of *LNCS*, pages 213–228. Springer, 2020.

[38] Michael A Specter, James Koppel, and Daniel Weitzner. The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in US Federal Elections, 2020. `https://internetpolicy.mit.edu/wp-content/uploads/2020/02/SecurityAnalysisOfVoatz_Public.pdf`.

[39] 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.

[40] Cynthia Sturton, Susmit Jha, Sanjit A. Seshia, and David A. Wagner. On voting machine design for verification and testability. In *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009*, pages 463–476. ACM, 2009.

A Building the voting device

The following boards were used for testing: DOIT DevKit v1, Wemos Lolin32, Heltek ESP32 Wifi Kit. Not all of these board have the same amount of pins and some pins are reserved for board-specific features, such as on-board displays. For example, Wemos Lolin32 boards generally have 26 pins, whereas DOIT DevKit v1 boards have 30.

A.1 Basic set-up

Wemos Lolin32 OLED and Heltec ESP32 Wifi Kit were used in the basic setup. Their main difference is the pin layout and crystal oscillator frequency, see Table 3. KY-040 rotary encoder is used for user input.

To connect it, two pins are used for rotation (CLK, DT) and one pin (SW) for clicking. Figure 5 presents the final set-up and Figure 3 the schematics of the layout.

Table 3: The pin layout for Wemos Lolin32 OLED and Heltec ESP32 Wifi Kit.

	OLED pins			KY-040 pins		
	SDA	SCL	RST	CLK	DT	SW
Wemos Lolin32 OLED (40 MHz)	5	4	-	16	13	14
Heltec ESP32 Wifi Kit (26 MHz)	14	13	16	19	21	23

A.2 Extended set-up

The extended set-up uses a 240 × 320 pixel ILI9341 touchscreen. In this set-up, the screen is connected to the board over serial peripheral interface (SPI), where the board is a master and the screen is a slave. Although ESP32 module integrates four SPI peripherals, only two of them (HSPI and VSPI) are open to users for general purpose. A single SPI bus consists of three lines and is shared among all connected sensors. Content on the screen (pixel array) is updated at each clock pulse (CLK line) through MOSI (Master Out Slave In) line.

SPI requires one or more CS (Chip Select) lines to allow the host to select a specific slave on the SPI bus to send or receive data. In addition, the driver

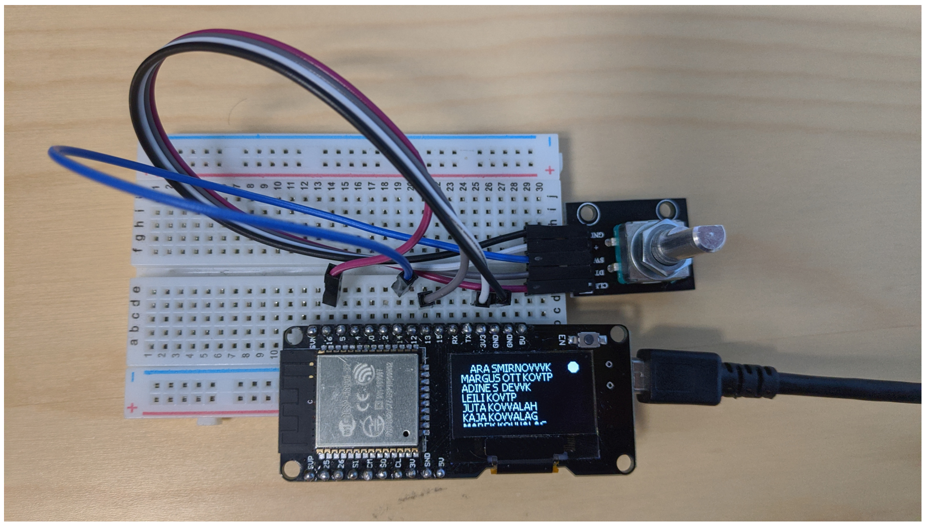


Figure 5: Set-up with rotary encoder and small display.

for ILI9341 uses three pins to control the display. D/C (Data/Command) pin is used to distinguish received bits as data or commands. RST (reset) and Lite (backlight) pins are required during initialization to reset the driver state and switch on backlighting. PIN configurations with DOIT DevKit v1 are as follows:

- SPI pins: CLK 14; MOSI 13; CS 15.
- ILI9341 pins: D/C 21; Lite 5; RST 16.
- Touch pins: Y+ 32(A4); Y- 25; X+ 26; X- 33(A5).

Figure 6 displays the final set-up and Figure 4 shows the connection schematics.

Similar to the display driver, one can use SPI bus to establish a communication to read touch responses with an additional touch controller (e.g. STMPE610). However, one can also directly read that data using four pins on the breakout board. By using analog-to-digital converter driver in ESP32 module, one can read voltage levels on specific pins and deduce touch coordinates. This functionality is provided by an open-source touchscreen library from Adafruit Industries that uses only 4 pins to read touch properties (x, y coordinates and pressure).[29] After screen calibration, one can port touch as an input source in LittlevGL.[30]

Screen calibration is used to convert touch events to coordinates. As simple linear scaling method gave sufficiently accurate results, we used scaling function

[29]`https://github.com/adafruit/Adafruit_TouchScreen/`
[30]`https://github.com/lvgl/lvgl`

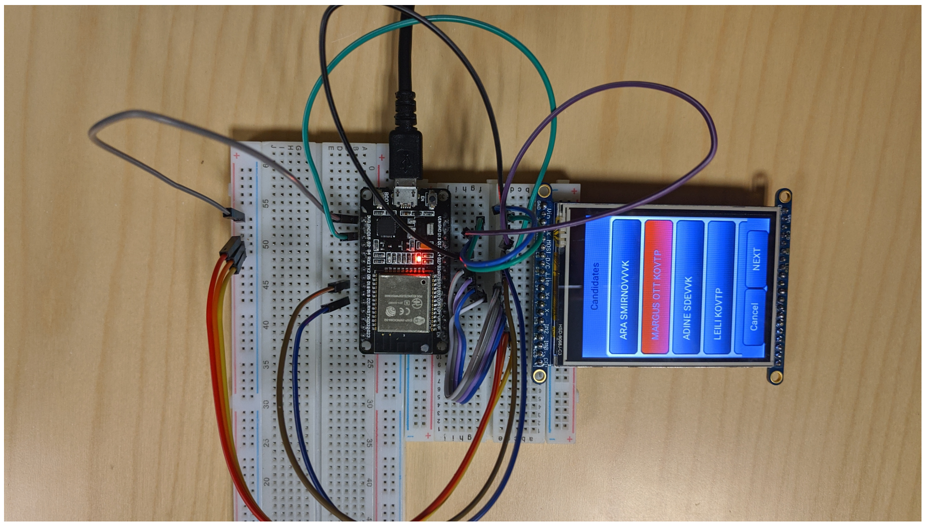


Figure 6: ILI9341 touchscreen and DOIT DevKit v1.

defined as

$$f(x) = (x - I_{min}) \cdot \frac{O_{max}}{I_{max} - I_{min}},$$

where x is voltage level on analog pin Y+ (X-) when the screen is touched. I_{min}, I_{max} are minimum and maximum values read from the touch sensor horizontally (vertically); and O_{max} is width (height) of the screen.

Acknowledgments

This paper has been supported by the Estonian Personal Research Grant number 920 and European Regional Development Fund through the grant number EU48684 and EXCITE centre of excellence. The authors would also like to thank Sven Heiberg for his support throughout the development and paper writing process, and Sergei Kuštšenko for help in proof-reading the code.