Reversing, Breaking, and Fixing the French Legislative Election E-Voting Protocol

Alexandre Debant, Lucca Hirschi

Reversing, Breaking, and Fixing the French Legislative Election E-Voting Protocol

Alexandre Debant
Université de Lorraine, Inria, CNRS, France

Lucca Hirschi
Université de Lorraine, Inria, CNRS, France

v2.1[†] — September 25, 2023

Abstract

We conduct a security analysis of the e-voting protocol used for the largest political election using e-voting in the world, the 2022 French legislative election for the citizens overseas. Due to a lack of system and threat model specifications, we built and contributed such specifications by studying the French legal framework and by reverse-engineering the code base accessible to the voters. Our analysis reveals that this protocol is affected by two design-level and implementation-level vulnerabilities. We show how those allow a standard voting server attacker and even more so a channel attacker to defeat the election integrity and ballot privacy due to 5 attack variants. We propose and discuss 5 fixes to prevent those attacks. Our specifications, the attacks, and the fixes were acknowledged by the relevant stakeholders during our responsible disclosure. They implemented our fixes to prevent our attacks for future elections. Beyond this protocol, we draw general lessons, recommendations, and open questions from this instructive experience where an e-voting protocol meets the real-world constraints of a large-scale, political election.

Contents

This work was partly supported by the following grants: ANR Chair IA ASAP (ANR-20-CHIA-0024) and ANR France 2030 project SVP (ANR-22-PECY-0006).

[†]A list of changes since the initial version can be found in Appendix G.

1 Introduction

Verifiability is a central goal in e-voting: it allows voters and auditors to verify the election result. Many e-voting protocols achieve (individual) verifiability for the voters thanks to a receipt bound to their ballots (*e.g.,* [5, 10, 15, 26]). Receipts allow to track the presence of ballots in the final bulletin board and election result and prevent a compromised or malicious voting server to drop or tamper with their cast ballots. In this paper, we ask the question how the French Legislative E-Voting Protocol (FLEP) does so by conducting a comprehensive security analysis.

The FLEP was the e-voting protocol used to organize the French legislative election for French residents overseas in June 2022 with 1.6 million eligible voters. This was the largest election (in terms of expressed votes) worldwide using e-voting. In total, to elect 11 deputies, more than 524k ballot have been cast and tallied[1]. The voters massively preferred using the FLEP (76%) over traditional paper-based voting (22.7%) or postal voting (0.3%) [1].

[1]This number includes the first and the second round of the election. To give a comparison, the second largest such election is the 2015 Australian state election with 280k expressed votes using iVote, that is 6% of the expressed votes, to elect 93 deputies.

As expected, the FLEP has high security ambitions. Some of those ambitions are related to lawful requirements and recommendations from all relevant regulatory bodies. To meet those requirements, notably that it remains secure under strong threats such as internal threats, the FLEP has undergone audits and the organizers have put in place an external third-party providing verification services operated by independent French researchers. Voters were encouraged to visit this web-service to verify the presence of their ballot (and receipt) in the ballot-box. For the researchers to independently develop such a tool, the vendor made public a rather incomplete specification partially describing the verifiability mechanisms used in this protocol.

In this paper, we answer the following question: Does the FLEP meet its goals? We reverse-engineered the FLEP and found flaws in the protocol and its implementation that could have been exploited to stealthily defeat ballot privacy of target voters and verifiability under a *voting server attacker* (compromised voting server) or under an even weaker *channel attacker*, that is: honest voting client and compromised plaintext channel client-server (an example of concrete scenario is a compromise of the voting server certificate). That is a strictly weaker attacker model than the standard compromised server threat model (which can be the result of internal threats).

Contributions. We contribute the following.

Reverse. We reverse engineered the obfuscated JavaScript program running in the FLEP voting clients. Doing this, fully passively in order to not alter the election, on a first version used during a large-scale test election and a second version during the main election, we were able to cross-reference information in order to fill the several critical gaps in the partial specification of the FLEP. We obtained this way a full specification of the voting client and the verifiability checks. We also studied the French lawful requirements and relevant recommendations to define a precise threat model specification that we argue is in line with the French legal framework and is also supported by the literature.

New Vulnerabilities. Analyzing this, we found two vulnerabilities that can be exploited by a channel attacker and even more so by a voting server attacker, which are both included in our threat model: (V_1) A channel attacker can break the bound between voters' ballots and their receipts due to an implementation flaw in the voting client. (V_2) We found that the sub-election identifier (associated to a consulate) is not correctly cryptographically bound to the ballot, but is to the receipt. Combined with (V_1), this allows a channel attacker to modify the sub-election identifier of a ballot.

Attacks. We show how a channel attacker could have exploited those vulnerabilities to stealthily carry out the following attacks (that is without leaving any evidence of the attacks): (A_1) By providing crafted receipts to voters, a channel attacker can selectively drop voters' ballots to defeat individual verifiability and modify the result of the election. (A_2) Choose the ballot that will be cast in place of the genuinely sent ballot while still providing a good looking and valid receipt. This attack is a variant of A_1 and is more effective at modifying the election result. (A_3) Defeat ballot privacy of target voters. We show how a channel attacker can learn how *target* voter(s) voted by moving around ballots from sub-election (*i.e.,* consulate) to another and observing the per-consulate result. We stress that this can be done while evading all possible detection and only assuming a channel attacker; in particular decryption trustees can all be trustworthy. (We also discuss 2 other attack variants.)

We do not claim that such attacks happened. We solely claim that a malicious or compromised channel or voting server had the technical ability to perform such attacks without leaving any evidence. We responsibly disclosed those attacks to all impacted and involved actors: the operator Europe and Foreign Affairs French Ministry (EFA Ministry), its institutional security advisor Agence nationale de la sécurité des systèmes d'information (ANSSI)[2], and the vendor Voxaly Docaposte. All of those 3 stakeholders have acknowledged and confirmed our attacks. We later disclosed our findings to the 3rd-party services supervisors. Note that a comprehensive risk analysis is out of the scope of this paper.

Fixes. We propose and discuss 5 countermeasures to those attacks. The EFA Ministry, ANSSI, and the vendor (Voxaly Docaposte) have confirmed to us that they have used some of our countermeasures to fix the FLEP. Already for the new elections that were organized in June 2023 (because the election was contested in some constituencies), FLEP has been partially fixed thanks to our work, as witnessed by the new specification [20]. We not not claim the absence of other attacks but only that the attacks we found were fixed.

Lessons. Designing a secure e-voting system is notoriously difficult. Flaws are regularly discovered on real-world protocols; *e.g.,* on those used in Estonia [29], Switzerland [24], Russia [22]. In the case of the FLEP, the protocol is claimed derived from the state-of-the-art academic protocol Belenios that is proven secure [15] (for the threat model we consider here) and is affected by none of those attacks. What went wrong?

To answer, we draw more general conclusions and lessons from this instructive experience where an academic protocol meets the real-world, challenging constraints of a large-scale and political election. Some of those translate to new open research questions. Those lessons are of a broader interest. In particular, we highlight the pitfalls in which the deployment of the FLEP fell, that are of general interest as they could have impacted the deployment of other state-of-the art protocols.

[2]The ANSSI is the *French National Agency for the Security of Information Systems* whose missions include cyber defense of state information systems and to provide advice and support to government and operators of critical national infrastructure.

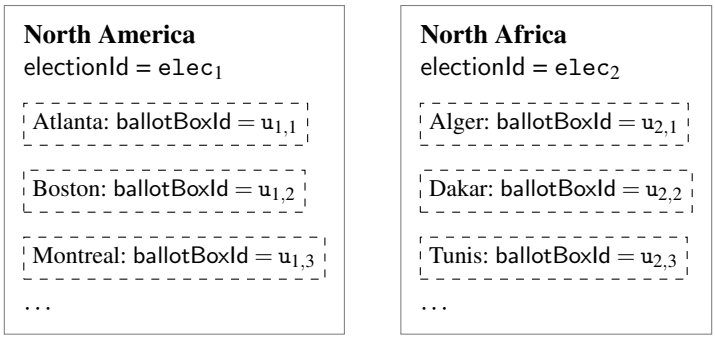


Figure 1: Organization of the French legislative elections (partial view). Each constituency (*e.g.,* North Africa) runs its own election to elect one deputy. The eligible voters of each constituency are grouped into consulates (*e.g.,* Alger) with their own ballot-box. Per-consulate results are also published.

Outline. In Section 2, we present the architecture, the security goals, and the threat model for the FLEP. In Section 3, we explain how we reverse-engineered the FLEP to build a system and threat model specification and describe the latter. In Section 4, we present the vulnerabilities and attacks we found and discuss our fixes. In Section 5, we draw more general lessons from this work, that go beyond the FLEP and conclude in Section 6.

2 Context

We first describe the context of the FLEP, its security goals, and threat model.

2.1 Architecture

The specification [19] published by the vendor is largely incomplete but provides useful information about the architecture and how the FLEP is deployed in practice.

Geographical Organization. French citizens overseas are gathered in 11 electoral regions (*e.g.,* north America, north Africa, etc.), which are *constituencies*. Each of the constituencies is itself split into several consular sub-regions which are *consulates* and typically represent a country or a city. Each constituency elects one deputy using the FLEP with a unique election identifier $\mathsf{electionId}$ cryptographically bound to each ballot. Each of those elections are organized in two rounds with their own identifier $\mathsf{roundId} \in \{1,2\}$. Additionally, the French law requires that the results are also published at the level of consulates. Therefore, to each consulate in a consistency is associated a ballot-box with a unique identifier $\mathsf{ballotBoxId}$. Figure 1 sums-up the organization.

Protocol Roles. Similarly to state-of-the art protocols such as Helios [5] or Belenios [15], the FLEP relies on different *roles* detailed next.

- **Voter:** they are the agents who will cast a vote. The FLEP assumes that voters own an email address and a cellphone number to receive login/password and confirmation codes before and during the election.
- **Voting device:** it is the device used by the voter to create and cast their vote. In the FLEP, the voting device is a JavaScript program provided by the voting server and executed in the voter's browser.
- **Voting server:** it is a server operated by the EFA Ministry whose purpose is to authenticate voters and collect all the ballots.
- **Decryption authorities:** they are the authorities who can decrypt the ballots. More precisely in the FLEP, they are 8 couples holder/deputy, and each of these 16 authorities own a share of a decryption key based on a 4-threshold encryption scheme, that is only a group of at least 4 authorities can collude to decrypt.[3] The 16 decryption authorities generate a unique public encryption key pk_E for all constituencies and their corresponding private keys $\mathsf{sk}_{\mathsf{E}1}, \ldots, \mathsf{sk}_{\mathsf{E}16}$.
- **3rd-party**: it is an external server operated by independent researchers and engineers from a French lab (LORIA) and research institutes (CNRS, INRIA), commissioned by the EFA Ministry. It provides to the voters some verification web-services [12] whose purpose is to ensure the integrity of the election. This independently developed and open-source program is available at [14].

[3] We will come back in Section 5 on this threshold and how the legal requirements for decryption could be better reflected in the cryptography.

2.2 Security Goals and Threat Model

To conduct a security analysis of the FLEP, one first needs precise security objectives and threat models. While the partial specification [19] fails to do so, there fortunately exist lawful requirements that the FLEP should comply with, notably the Code électoral [21] (*i.e.,* the French law governing elections) and the recommendations enacted by the Commission nationale de l'informatique et des libertés (CNIL) [30][4], which are a list of requirements that such a system is expected to meet (even if there is no legal obligation).

We have studied this legal framework [21, 30] and derived security goals the FLEP should achieve and under which threat model. Our list of objectives is not exhaustive but we focus here on those that are relevant for our work. We refer the reader to Appendix A for detailed explanations about our study of this framework and how we derived the security goals and threat models that we summarize next.

Ballot Privacy. The Code électoral [21, R176-3-9] and the CNIL requirements [30, SO 1-04,1-07] agree on the fact that the FLEP must ensure the *confidentiality of the votes*.

Verifiability. Second, the FLEP must ensure the integrity of the election outcome and notably **individual verifiability** [21, R176-3-9], [30, SO 2-07,3-02], that is each voter should be able to verify that their ballot has been added into the ballot-box; *i.e.,* it was not dropped or tampered with.

For instance, the requirement [21, R176-3-9] requires that "the voter is provided with a digital receipt allowing them to verify online that their vote has been taken into account". As we shall see, the FLEP uses some hashed values over the cast ballot and some meta-data as well as a signature as *digital receipt*. The voter can verify online this receipt at the voting server or the 3rd-party server, the latter independent 3rd-party verification option must exist to comply with [30, SO 3-02].

Threat Model. We shall define a threat model that we argue is in line with the legal framework of the FLEP.

As detailed in Appendix A, we derive from the CNIL recommendations [30, Security level 3, SO 3-02] that the FLEP must guarantee the integrity of the election even under a compromised voting server, which also motivates the requirement to rely on an independent, 3rd-party verification service. Most of the attacks we shall present actually make weaker assumptions, *i.e.,* assume an even weaker attacker. Indeed, instead of considering a (possibly) compromised voting server, we shall assume a compromised communication plaintext channel between the voters and the server, we call such an attacker a *channel attacker*. More precisely, the *channel attacker* can intercept and inject (plaintext) messages in-between voters under attack and the voting server but has not necessarily access to the voting server internals, such as its databases, the signing sheet, the authentication material, the log files, etc.

Remark 1 (Examples of channel attackers)**.** *A channel attacker can obviously be realized by compromising the voting server. Indeed, even if the converse is wrong in general, a voting server attacker is also a channel attacker.*

More interestingly, it can be realized by compromising parts of the network server infrastructure. Specific operational countermeasures are usually deployed to protect the critically important server(s) on which the election runs. However, the server(s) are connected to the Internet through a more complex and shared network infrastructure which may appear as an Achilles heel. The TLS certificates shared across the infrastructure might be less protected and shared across more devices such as TLS middle-boxes that could be deployed to monitor the plaintext traffic to mitigate e.g., DDoS attacks. This, unfortunately standard, solution would undermine the security of the communication channels voters-voting server by increasing the attack surface and the risk of man-in-the-middle *attacks yielding a channel attacker.*

Finally, a channel attacker could also be the result of a compromise of the network infrastructure on the voter side. For similar reasons, a voter's company may monitor the plaintext Internet traffic to protect its employees and its assets. Unfortunately, this would, again, introduce a man-in-the-middle *in the communication channel between the voter and the server. Voters have not been made aware of this threat in the context of this election.*

Therefore, a channel attacker is an interesting threat model in itself as it is strictly weaker than the server attacker and can be realistic in some scenarios.

Table 1 summarizes the assumed trustworthiness of all roles for the different properties (the less trustworthy parties there are, better is the protocol). It also compares this with the threat model under which the Belenios protocol (and other state-of-the-art protocols such as Helios [5]) were proven secure. (An honest public bulletin board is usually assumed, which can be achieved by an honest 3rd-party for comparison.) It shows that our attacks break the FLEP under an (even weaker) attacker than the standard one for which Belenios was proven secure [15], despite FLEP being derived from Belenios (according to [19]).

Remark 2. *Assuming the voting client as a trustworthy component can be considered too strong an assumption, even more so due to the latter being a JavaScript program served by the voting server and executed in the voter's browser. Some e-voting systems*

[4]The CNIL (*National Commission on Informatics and Liberty* in English) is an independent French administrative regulatory body whose mission is to ensure that data privacy law is applied.

Security Goal	Voter	Voting Device	Com. Channel	Voting Server	Dec. Auth.	3rd-party
Our attacks on the FLEP falsify the properties under the threat model:						
Verifiability	✓	✓	✗	✓	✓	✓
Ballot privacy	✓	✓	✗	✓	✓	✓
State-of-the-art protocols such as Belenios are secure under:						
Verifiability	✓	✓	✗	✗	✗	✓
Ballot privacy	✓	✓	✗	✗	✓	✓

✓= assumed trustworthy, ✗= can be untrustworthy

Table 1: Comparison of the threat models under which FLEP is breached and the strictly stronger threat model under which state-of-the-art protocols such as Belenios are secure [15].

such as the IVXV [25], Helios [5], or the Swiss Post protocol [34], aim at ensuring the well-known cast-as-intended *property, that is ensuring individual verifiability under a compromised voting device that could try to modify the intended vote. The FLEP protocol has obviously not been designed to protect the voters against such a threat. Therefore, to conduct a fair security analysis, we decided to assume the voting client as a trustworthy component. We discuss in Section 5 and Appendix D.5 how this assumption can be made realistic in practice with the notion of "*universal integrity checks*" where anyone can check the JavaScript integrity to detect malicious voting servers serving malicious code.*

3 Reverse the Protocol

[19] only provides a partial specification, that is so incomplete that the attacks we shall present could not be even described[5]. Indeed, this document focuses on the expected format of some verifiability-related messages (such as the receipts) but omits to specify how exactly they are computed, exchanged, and which checks are performed upon reception. Moreover, a bigger picture of the protocol is completely missing.

By reversing the JavaScript voting client code, studying and cross-referencing different sources, we built a complete description of the FLEP voting client and all its interactions with the server as well as some important server-side components (handling of errors, verifiability checks). All the impacted and involved actors (Voxaly Docaposte, ANSSI, EFA Ministry) have acknowledged our system specification is correct.

Note that even though a specification of the 3rd-party tool was missing too, we have been able to determine the checks it performed based on an informal description of the service [12], open discussions with the researchers, and an inspection of the source code.

3.1 Reverse Methodology

We describe next how we overcame the lack of complete specification by reversing the obfuscated voting client.

Obtaining data. During the election, we collected all the web browser's interactions with the voting server throughout the different steps of the protocol thanks to a few eligible voters[6]. These are gathered into HTTP Archive format (HAR) files that can be easily generated by major browsers. We collected for one typical voter's journey 15 JavaScript files, 4 HTML files, 4 plain data exchanges, etc.

Reverse engineering and data cross-referencing. We first recollected the overall flow of messages by analyzing all the POST and GET requests in a typical voter's journey. (We give detailed descriptions thereof in Appendix C.1.) We cross-referenced this with the description of the voter's journey published on the government website to help voters with the voting process. We obtained this way a clear big picture of the overall flow of messages of the FLEP.

Some of the fields of the GET and POST requests can be related to the partial specification [19] but many are omitted or not fully described. Moreover, even for those fields that were specified, we needed a better understanding about how they are computed and how they are checked. For this, we had to investigate the JavaScript programs that produce and check those

[5]Based on our discussions with the stakeholders, Voxaly Docaposte published in February 2023 a new version of the specification [20] which includes a more detailed protocol specification with some countermeasures to our attacks we discuss in Section 4.

[6]The voters who sent us the collected data, who are computer scientist colleagues, were informed about our research and about the content of those logs. They gave us their informed consent.

fields. Excluding all-purpose library files (such as `jquery-3.1.1.msin.js` or Captcha-related files), there are 4 JavaScript files that are specific to the FLEP implementing its core logic (`election.bundle.js`, `loria.bundle.js`, `app.bundle.js`, and `verifiabilite.bundle.js`) totaling 15574 LoC when de-minimized with js-beautify[7].

Those files are obfuscated: they were processed using obfuscation techniques such as function and variables renaming, or control flow modifications in order to make reverse engineering more complex, as is standard with web-development. In our case, some variable names were not obfuscated, in particular request fields were not. This way, we were able to locate part of the JavaScript code manipulating those fields. However, the control flow is so obfuscated that it makes it very hard to keep track where those fields are flowing. See for example Listing 2 from line 1 to 11 (line 12-17 turned out to contain the core logic manipulating the receipt).

Fortunately, we obtained a previous version of these JavaScript files used during the first large-scale, in-the-wild test campaign of the system conducted in September 2021 [2]. Interestingly enough, the code and the protocol for this test phase was a bit different and most importantly for us, the code was less obfuscated. In particular, the control flow was less obfuscated. We thus decided to reconcile the two code bases and cross-reference some of the most interesting function implementations. We improved our understanding of the overall logic of the code by investigating the test phase code base and then by cross-checking

[7] https://www.npmjs.com/package/js-beautify

```
1  navclientApp.controller("PageVoteController", ["$scope", "$http", "$location", "$timeout", "
       breadCrumbService",
2  function(e, t, i, n, a) { // HashClient core logic
3    e.vote = function() {
4      if (data.param.signatureEnabled && !e.aVote) {
5        e.aVote = !0, e.erreurHashVerification = !1;
6        var i = forge.md.sha256.create();
7        i.update(e.bulletinCrypte + data.election.ordre + data.param.electeurEtOrdre);
8        var n = i.digest().toHex(),
9          a = function(e) {
10 //           [7 lines omitted]
11         }(n),
12         o = data.election.ordre + "&" + n + a;
13       sessionStorage.setItem("HashClient", o);
```

Listing 1: Test Phase, `scripts.js`

```
1  function(e, t, n) {
2  // [1729 lines omitted, indentations were removed]
3  function ot(e) {
4  // [73 lines omitted]
5  function v() {
6    return (v = Pe()(Re.a.mark((function t() {
7      var n, r, a, l, u, c, s;
8      return Re.a.wrap((function(t) {
9        for (;;) switch (t.prev = t.next) {
10         case 0:
11 //           [9 lines omitted]
12         case 3: // HashClient core logic
13           return (n = new jsSHA("SHA-256", "TEXT")).update(o.bulletinCrypte + f.idTour + d.ordre + f.
                 electeurEtOrdre),
14           r = n.getHash("HEX"),
15           (a = new jsSHA("SHA-256", "TEXT")).update(o.bulletinCrypte + o.voteSignature),
16           l = a.getHash("HEX"),
17           u = f.idTour + "&" + d.ordre + "&" + r + y(r),
18           sessionStorage.setItem("HashClient", u),
```

Listing 2: Production phase `app.bundle.js`

Figure 2: Snippets of code from test phase and production phase relevant to the computation of the `HashClient` stored in the web browser session storage (*i.e.,* persistent storage). Comments were added by us. The control flow of the test phase (Listing 1) is much simpler and easier to reverse (When is this piece of code triggered? What happens next?), as opposed to the production phase code (Listing 2). Conversely, the production phase is the version that matters and some message contents have changed. Therefore, we had to cross-reference both code bases.

Name	Expected value
Ballot	$b := (\{v\}_{\mathsf{pk_E}}, \pi)$
Hash value	$h := \mathsf{hash}(b \Vert \mathsf{roundId} \Vert \mathsf{electionId} \Vert \mathsf{ballotBoxId})$
Ballot reference	$H := \mathsf{roundId} \Vert \mathsf{electionId} \Vert h$
Activation hash	$h_{\mathsf{code}} := \mathsf{hash}(b, \mathsf{code}_{\mathsf{activ}})$
Ballot fingerprint	$hb := \mathsf{hash}(b)$
Seal	$\mathsf{cSU} := \mathsf{infoSU} \Vert \mathsf{sign}_{\mathsf{sk_S}}(\mathsf{hash}(\mathsf{infoSU})) \Vert \mathsf{pk_S}$
	$\mathsf{infoSU} := \mathsf{roundId} \Vert \mathsf{electionId} \Vert \mathsf{electionName} \Vert \mathsf{ballotBoxId} \Vert hb^{s_5}$

Table 2: Main cryptographic values involved in the FLEP.

with the production code base. An example of such a side-by-side comparison is depicted in Figure 2.

Reverse engineering checks and errors. For obvious reasons, we have forbidden ourselves to carry out active attacks against voting servers. Therefore, we limited ourselves to passive sniffing of the exchanged messages. This makes it hard to understand what happens when something goes wrong (since this never happened for the sessions for which we collected data). Some checks are carried out in the voting client and we were able to reverse them. But the others are carried out in the server side. For those, our logs were not helpful at first sight. Fortunately, some error messages are built-in in HTML pages in hidden `div` environment. By locating the JavaScript logic, we were able to partially understand the conditions under which those errors are displayed upon reception of some messages from the voting server.

The following description of the system may miss actions executed by an honest voting server, but as we shall see, it is precise enough to claim that our attacks are valid independently of those unspecified components. Indeed, our attacks rely on sending data to the server that are indistinguishable from its point of view from data honest voters would produce.

3.2 Reversed Specification

We now present the result of our reverse: a full description of the FLEP. We shall specify all the actions executed by the voter, the voting client, and the voting server at each step. Unlike [19], we also explicit how messages are exchanged and computed. Those steps are summarized in Figure 3.

Step 1: The voter browses to the election website URL and connects to the voting server and receives an HTML document displaying the login page. The voter authenticates themselves using a login and password they received before the election through two different channels, the login by email and the password by SMS. In addition to the authentication data, the voting client also sends to the voting server client_info (some meta-data about the voting client) and b_{test} which is a dummy ballot encrypted with a dummy election public key that serves as sanity check that the voting client will be able to correctly compute the real ballot at step 4.

Step 2: The voting client receives an HTML document displaying the different candidates $\mathcal{V}$ the voter can choose. This document also contains some hidden identifiers such as the election identifier electionId or the ballotBoxId and a per-session unique identifier `tokenId`. The voter chooses one candidate $v \in \mathcal{V}$ for whom they want to vote.

Step 3: The voter clicks to confirm their choice.

Step 4: The voting client sends a request to the voting server to generate an *activation code*, acting as a second authentication factor. The voting server generates such an activation code $\mathsf{code}_{\mathsf{activ}}$ (made of 6 random digits) and sends it to the voter using a side-channel (*i.e.,* by email). The voter receives this code and enters it in the voting client and clicks on the "Vote" button. The voting client then computes some cryptographic material explained next and summarized in Table 2.

• First, the *ballot* $b := (c, \pi)$ that is made of the encrypted vote v ($c := \{v\}_{pkE}$) with the election public key (pkE) along with a Zero-Knowledge Proof (ZKP) (π) proving that v is a legitimate choice of candidate, *i.e.,* $v \in \mathcal{V}$. This proofs is also bound to `tokenId` and electionId received at Step 2.

• The voting client also computes the *hash value*:

$$h := \mathsf{hash}(b \Vert \mathsf{roundId} \Vert \mathsf{electionId} \Vert \mathsf{ballotBoxId})$$

where $\Vert$ is a delimiter, roundId is the round of the election (1 or 2), electionId is the election identifier, and ballotBoxId is a per-ballot-box (hence per-consulate) unique identifier.

• The *ballot reference* (omitting correction codes):

$$H := \mathsf{roundId} \Vert \mathsf{electionId} \Vert h$$

is computed and exchanged at different steps of the protocol, we specifically note H^c to refer to the *ballot reference* that is

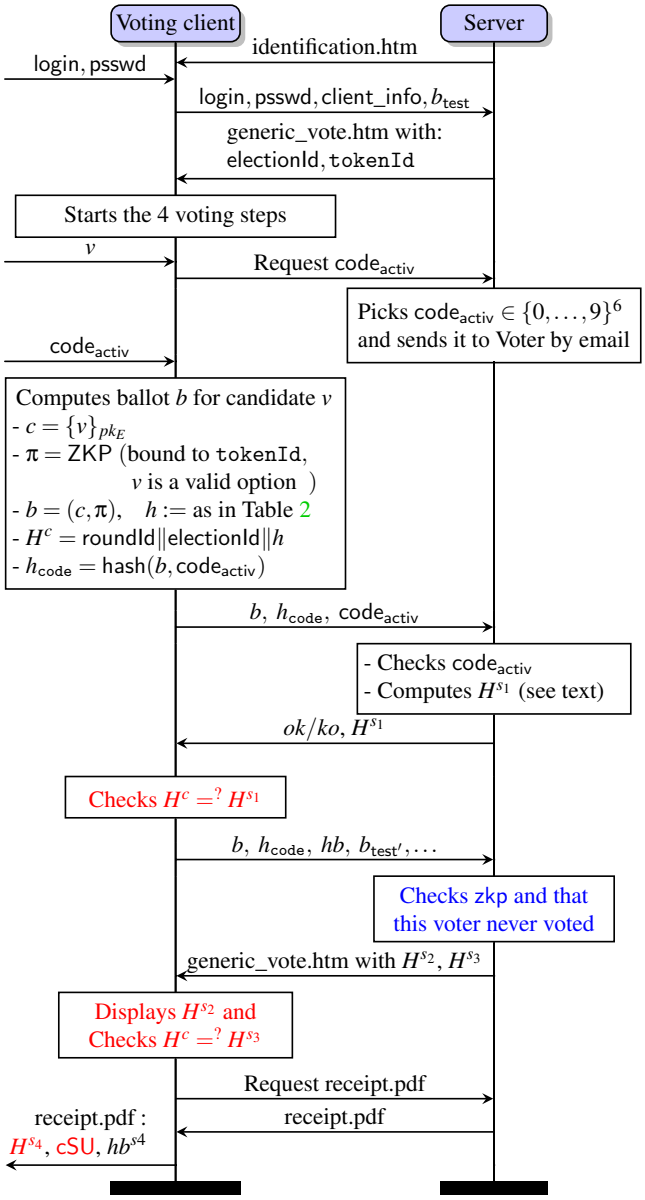


Figure 3: Description of the 2022 French Legislative Election Protocol. Arrows on the left correspond to interactions with the human voter. Cryptographic messages are specified in Table 2. Actions in red will be relevant for the vulnerabilities presented in Section 4. Actions in blue, and only those, are elements obtained during discussions with the stakeholders. A more detailed diagram is given in Appendix C.

computed by the voting client and stored in the SessionStorage of the voting client browser[8].

• The *activation hash value* $h_{\text{code}} := \text{hash}(b, \text{code}_{\text{activ}})$. This value was not described at all in the specification [19] and was supposed to act as a proof of knowledge of the activation code $\text{code}_{\text{activ}}$, bound to the ballot.[9] That said, the precise role of h_{code} is unimportant for the rest of the presentation.

The values b, h_{code}, and $\text{code}_{\text{activ}}$ are then sent to the voting server. The voting server then verifies the validity of the activation code $\text{code}_{\text{activ}}$, recomputes the *ballot reference* H, denoted by H^{s_1}; indeed H^c is not sent to the voting server. The response to the

[8]The SessionStorage is a storage local to the browser and associated to the current tab. It allows to store session-specific data that persist page reloads and page redirections. See, for example, the documentation for Firefox [3].

[9]We note that this is completely inefficient since the activation code $\text{code}_{\text{activ}}$ is actually sent in clear text along h_{code} (with the ballot b). According to the vendor, this was a mistake and will be fixed. Once fixed, h_{code} provides such a proof, even though $\text{code}_{\text{activ}}$ can be quite easily brute-forced given h_{code} and b.

above request is $(ko, \_)$ if the activation code was invalid and (ok, H^{s_1}) otherwise. Upon reception of the response, the voting client verifies that the response equals (ok, H^c). Therefore, H^{s_1} must be equal to H^c. This test provides the voting client evidence that it executes the protocol in the same context as the one of the voting server.

Finally, if those tests succeed, then the voting client computes the *ballot fingerprint*: $hb := \mathsf{hash}(b)$. The ballot b is again sent to the voting server, along with hb and $h_{\texttt{code}}$.

At this point, the voting server verifies that the voter never voted before (revoting is forbidden in the FLEP) and that the ballot ZKP π is valid. If so, the voting server stores the ballot b in the ballot-box $\mathsf{ballotBoxId}$ associated to the voter (one per consulate). It also adds the voter to the *signing sheet* containing all the voters who voted so far. Finally, it computes and stores a *seal* cSU associated to this ballot, that is a signature over the ballot and some metadata that will be part of the receipt provided to the voter in the final step (we specify the seal cSU below).

Step 5: Finally, the voting client receives from the voting server an HTML document that displays the *ballot reference H* and that asks the voter to click a link to download the PDF receipt. Voters are also encouraged to click a link to verify that their ballot is indeed included in the ballot-box.

Two occurrences of the *ballot reference H* are present in this HTML document, which is generated and sent by the voting server. We thus denote those two occurrences with respectively H^{s_2} and H^{s_3}. The first occurrence H^{s_2} appears in a HTML tag $< \texttt{pclass} = \texttt{"recepisse-code"} >$ and corresponds to the value being displayed to the voter. The second occurrence H^{s_3} appears in a JavaScript script embedded in the page that tests that H^{s_3} equals the *ballot reference* stored in the voting client SessionStorage, *i.e.,* H^c. Therefore, $H^{s_3} = H^c$ or an error message is displayed.

PDF receipt: The very last step occurs when the voter clicks the link to download the PDF receipt from the previous page, which replaces the previous page (which cannot be loaded any more). The downloaded PDF receipt contains the *ballot reference H*, the seal cSU, and the *ballot fingerprint hb*. Despite those values being (partially) specified in [19], the values displayed in the PDF receipt are never checked by the voting client and could differ from the expected, specified values. Therefore, we note H^{s_4} the *ballot reference* and hb^{s_4} the *ballot fingerprint* that are displayed in the PDF receipt.

The seal cSU should be computed by the voting server (already at step 4) as follows: $\mathsf{cSU} := \mathsf{infoSU} \| \sigma \| \mathsf{pk_S}$ where infoSU is defined as:

$$\mathsf{roundId} \| \mathsf{electionId} \| \mathsf{electionName} \| \mathsf{ballotBoxId} \| hb^{s_5}$$

and $\mathsf{electionName}$ is the name of the election, hb^{s_5} is supposed to be the *ballot fingerprint*, and σ is a digital signature with the server's signature private key $\mathsf{sk_S}$ (associated to the signing verification key $\mathsf{pk_S}$) computed as follows: $\sigma = \mathsf{sign}_{\mathsf{sk_S}}(\mathsf{hash}(\mathsf{infoSU}))$.

Decryption authorities: At the end of the election, a tally ceremony involving a threshold of the decryption trustees occurs. They collaborate to decrypt an homomorphic aggregate for each ballot-box, produce ZKPs of correct decryptions, and announce the election results.

3rd-party role: The 3rd-party contributes to both the individual and universal verifiability. To this aim, the EFA Ministry sends it the results of the tally (*i.e.,* the ballot-boxes, ZKPs, and results). First, for checking universal verifiability, the 3rd-party checks the validity of the ZKPs of correct decryption. At the constituency level, it checks that those proofs are valid *w.r.t.* official results publicly published onto the EFA Ministry website. Moreover, it checks that all the ballots included in the ballot-boxes are well-formed, *i.e.,* that their ZKPs are all valid. For allowing individual verifiability checks, the 3rd-party proposes a web service to the voters who can check that the seal of their receipts contains a legitimate signature of the voting server. Moreover, once the election is over and the 3rd-party has received the ballot-boxes, the 3rd-party also verifies that the seal corresponds to a legitimate ballot[10]. After the protest period, *i.e.,* ten days following the results announcement, the 3rd-party publishes a report [12] containing a summary of all their verifications.

4 Vulnerabilities, Attacks, and Fixes

We present the two vulnerabilities we found (Section 4.1). We then show how these vulnerabilities can be exploited to break verifiability and the integrity of the election (Section 4.2) and the confidentiality of target voters' votes (Section 4.3). We also propose fixes to those attacks, some of them are or will be deployed by the FLEP stakeholders. We summarize all the attacks we found in Table 3. We also provide the fixes chosen by ANSSI, EFA Ministry, and Voxaly Docaposte and already implemented for the elections organized in 2023.

[10] In 2022, voters had to use this service once the election was over, and not before, to obtain this guarantee. In 2023, based on our recommendation (see Appendix E.3.1), the 3rd-party decided to log all the seals received during the voting phase to do this check as soon as it becomes doable.

Name	Attack on	Threat model	Impact	Fix that have been deployed in 2023
Replace	Indi. Verif.	Channel att.	Replace any cast ballot	Fix 1: display H^c
Drop	Indi. Verif.	Channel att.	Drop any cast ballot	
Swap_H	Ballot priv.	Voting server att., some voters collude	Learn any target voter's vote	Fix 3: add ballotBoxId to the ZKP
Swap_b	Ballot priv.	Channel att., some voters collude	Learn any target voter's vote	
Swap_{ID}	Ballot priv.	Voting server att., some voters collude	Learn any target voter's vote	Fix 3 + Fix 5: display ballotBoxId

Table 3: Summary of the attacks found. The last column presents the fixes that have been chosen by the stakeholders and implemented for the elections organized in 2023.

4.1 Vulnerabilities

When reversing the specification, we uncovered 2 critical vulnerabilities that could be exploited by a channel attacker (*i.e.,* an attacker controlling the plaintext communication channel) and even more so by a voting server attacker (*i.e.,* a compromised voting server). As we shall see, they impact the integrity of the election and the confidentiality of the votes.

We stress that our reversed specification detailed in Section 3 was necessary to identify those vulnerabilities that could not even be described within the limited framework of [19].

V1: Lack of binding of receipts with their ballots. In the FLEP, the integrity of the election is guaranteed by the use of receipts: voters can send their receipt (the ballot reference H or/and the seal cSU) to a server (the voting server or the 3rd-party) to verify that their ballots have been added in the ballot-box. Moreover, the 3rd-party ensures, by verifying the decryption ZKPs, that the result of the election corresponds to the content of the ballot-boxes. Thanks to these two checks, an attacker should not be able to tamper with the cast ballots and with the election result.

A subtlety we noticed in the JavaScript code of the FLEP voting client is that the ballot references that are displayed to the voter by the voting client (H^{s_2} and H^{s_4}) for later checks are not necessarily the same as the one it computed for the ballot produced by the voting client (H^c). This is the reason why we named those references differently, even though they refer to values that are expected to be equal (*i.e.,* equal to H^c computed by the voting client) thanks to various consistency checks (shown in red in Figure 3).

Unfortunately, these checks are flawed in that, among the 4 references received from the voting server, H^{s_1}, H^{s_2}, H^{s_3}, and H^{s_4}, they only ensure that:

$$H^{s_1} = H^c \quad \text{and} \quad H^{s_3} = H^c.$$

There is no guarantee about the values H^{s_2} and H^{s_4}, which are the only references visible by the voters, respectively in the last web page and in the PDF receipt (both are depicted in Appendix C.2). A channel attacker can take advantage of this implementation flaw to falsify the verifiability of the FLEP and thus modify the result of the election as explained in Section 4.2. We stress that the human voters themselves are unable to recompute the genuine, honest value of H (that is H^c) without an expert and technical knowledge. Indeed, for instance, they never learn the value of their ballot b.

V2: Malleability of ballot-box identifiers. As presented in Section 2.1, the FLEP is deployed in a complex environment with multiple elections and ballot-boxes. In the partial specification of the system [19], Voxaly Docaposte seems to be aware of this complexity: "*The election identifier [*electionId*] [...] will be added in the [ZK] proofs associated to the ballot. This allows to detect if a ballot has been moved from a ballot-box to another*".

Unfortunately, we found out that this statement is incorrect. Indeed, the election identifier electionId does not identify a ballot-box (associated to a consulate) but only an election (associated to a constituency). The ballots are thus cryptographically bound to an election but not to a ballot-box whose identifier is not included in the ZKP context of the ballot.

Interestingly, the ballot reference H *does* cryptographically bind the ballot to the ballot-box identifier ballotBoxId. However, as we shall see, this ballot reference can be recomputed by an attacker for a different ballotBoxId. As we shall see in Section 4.3, a channel attacker can use this and the previous vulnerability (V1) to move around ballots across different ballot-boxes and attack ballot privacy.

4.2 Attacking and Fixing Verifiability

We now present how the first vulnerability can be exploited to break verifiability and thus the integrity of the election. We also propose and discuss fixes.

4.2.1 Attacking Verifiability

The first vulnerability can be exploited to falsify the individual verifiability of the system under a channel attacker. An attacker who controls the communication channels can stealthily (1) drop ballots, and (2) replace them by ballots of his choice as explained in the two attack descriptions below.

Attack [Replace]: This scenario assumes a channel attacker and requires at least two voters who vote in the same consulate: Bob, who will suffer from the attack, and, Alice, an arbitrary voter. In the following, we re-use all the notations introduced in Section 3.2 and Figure 3. We assume that all the variables are indexed by 1 for Alice's vote, and 2 for Bob's. The attack proceeds as follows:

Step a: Alice casts her vote as expected. In this first step, the attacker executes honestly, *i.e.,* $H_1^{s_i} = H_1^c$ for all $i \in \{1,\ldots,4\}$ and σ_1 is honestly computed. Hence, all the checks that Alice can do to be sure that her ballot b_1 has been included in the ballot-box succeed.

Step b: Bob follows the protocol but the channel attacker intercepts all messages after the creation of the ballot b_2 and computes by itself the responses that are normally sent by the voting server to the voting client as follows: $H_2^{s_1} = H_2^{s_3} = H_2^c$ but $H_2^{s_2} = H_2^{s_4} = H_1^c$ and $\sigma_2 = \sigma_1$. That is the attacker provides a receipt that refers to Alice's ballot. We do not assume that the attacker knows the signing private key $\mathsf{sk_S}$ to compute σ_2 since the attacker can simply replay the values obtained at Step a. The value H_2^c can be computed by the attacker since it is only made of public data (e.g. electionId and ballotBoxId) or data sent by Bob, such as b_2.

Step c: Because a ballot is not cryptographically bound to a voter, the attacker can forge a ballot $b_{\mathtt{att}} = (\{v'\}_{\mathsf{pk_E}}, \mathsf{zkp}')$ for $v' \in \mathcal{V}$ of his choice and replace each occurrence of b_2 by $b_{\mathtt{att}}$ in Bob's messages towards the voting server. More precisely, the attacker sends $b_{\mathtt{att}}$, $\mathsf{hash}(b_{\mathtt{att}}, \mathsf{code}_{\mathsf{activ}})$ and $\mathsf{hash}(b_{\mathtt{att}})$ instead of b_2, $h_{\mathtt{code}}$, and $\mathsf{hash}(b_2)$. The voting server then registers the adversarially-chosen ballot $b_{\mathtt{att}}$ in the name of Bob.

Even if, at Step b, Bob receives inconsistent data (*e.g.,* $H_2^{s_2} \neq H_2^c$), these are not detected by the checks performed by the voting client and those differences are invisible to Bob himself. (This seems similar to the clash attacks of [27] but our attack does not rely on a compromised voting client to make it compute a clashing ballot.) Moreover, Bob can use the web services (provided by the voting server or the 3rd-party) to be convinced that "his" ballot has been added to the ballot-box. Indeed, Bob got Alice's receipt which corresponds to a ballot added in the ballot-box at Step a. Finally, let us explain how the attacker can make sure Bob does not detect the replacement of his ballot in the very unlikely situation where his original ballot was the only vote for a particular choice $v \in \mathcal{V}$ in this ballot-box (since the tally will reveal v got no vote). It suffices for the attacker to make sure each of the options $v \in \mathcal{V}$ gets at least one vote: either by assuming the existence (*e.g.,* large consulates) or knowing other voters casting such ballots (*e.g.,* accomplices), or by performing the above attack on $|\mathcal{V}|$ voters to make them cast all options. As a result, all the checks Bob was suggested, instructed, or able to do succeed despite his ballot b_2 has been dropped and replaced by the attacker ballot $b_{\mathtt{att}}$.

Therefore, at the end of this scenario, nobody can detect the removal and replacement of Bob's ballot and yet, the election result will not include Bob's ballot b_2 and include the attacker ballot $b_{\mathtt{att}}$ instead. Alice and Bob are convinced that their ballots have been counted and the voting or the 3rd-party server will always receive consistent data. Bob is cheated by the fact that the receipt he obtained actually points to Alice's ballot, which does exist in the ballot-box. Note that a channel attacker can repeat this to attack an arbitrary number of voters acting as Bob, always using the same Alice's data. (Moreover, we describe in Appendix B.1 a weaker variant [Drop] of this attack that only drops Bob's ballot instead of replacing it.)

Impact: An attacker who compromises the communication channel (or the voting server) can significantly modify the outcome of the election by dropping and replacing ballots, hence falsifying the individual verifiability property.

Remark 3. *Interestingly, [Replace] is no longer possible if FLEP used the ballot format of Belenios, since this includes voters' signatures. Therefore, the removal of signatures from Belenios to FLEP –as a means to comply with CNIL recommendations (ballots should be anonymous)– without compensating this loss with another security mechanism introduced a weakness in the protocol.*

4.2.2 Fixing Verifiability

We propose 2 fixes to prevent such attacks. The first one is easy to implement but makes the voter's journey and verification tasks more complex, while the second would actually simplify them but is more complex to implement.

Fix 1: A first approach to fix the verifiability attacks is to display H^c instead of H^{s_2} on the last web page of the user interface (step 5). If this was done, a conscientious voter would be able to compare this reference to the one printed in their receipt to enforce the consistency $H^{s_1} = H^{s_2} = H^{s_3} = H^{s_4} = H^c$. Finally, the voter can then use the 3rd-party web service to check the presence of their ballot in the ballot-box with confidence. This solution would fix Vulnerability 1 as well as the two verifiability attacks [Replace] and [Drop]. However, it complicates further the voter's tasks, which are already quite complex (see Section 5).

Fix 2: A second, better fix is to make the voting client generate the PDF receipt. To do so, the voting server would still need to send the signature σ for the voting client to compute the seal cSU. The voting client must verify the validity of this signature (with respect to $\mathsf{pk_S}$) and its content before generating the PDF receipt by itself. We prefer this solution as it does not require extra voters' checks. Moreover, it allows to check the signature of the seal cSU before displaying the PDF receipt, allowing to detect potential forgery or voting server misbehavior as early as possible and making the voting server accountable for potential misbehavior detected later. The main drawback is that it requires to import an external library to generate PDFs or to use a static PDF file that is then dynamically filled with missing cryptographic data.

Remark 4. *The stakeholders chose to implement Fix 1 for elections in June 2023. Fix 2 seems to be their ultimate solution for a future version of the system.*

Remark 5 (On Fix 1)**.** *The version 2 of the specification [20] (from February 2023) shows that the Fix 1 was not implemented as is. We regret that the chosen implementation does not fully prevent the vulnerability V1. Indeed, H^c is now displayed on the last web page of the user interface (step 5) and the voting client now checks that $H^c = H^{s_3}$. Very surpinsgly, the vendors also decided to display H^{s_3} on the same page and even to instruct the voters to visually "check that the receipt H^c [computed by the voting client] corresponds to the receipt actually inserted in the ballot-box H^{s_3}". Asking the voters to visually check $H^c = H^{s_3}$ is useless given that this test is performed by the voting client. More importantly, this gives a false sense of security as it gives the impression no further checks are needed. The voters must actually check that the displayed receipt (H^c) corresponds to the one printed on the PDF receipt (H^{s_4}) and then use it to perform the usual individual verifiability check, preferably at the 3^{rd}-party website.*

In conclusion, the implemented fix allows a very cautious voter to detect our attacks (by comparing H^c and H^{s_4}). However, it also gives non-necessary additional tasks to the voters and fails to instruct them to perform the only useful check.

4.3 Attacking and Fixing Ballot Privacy

The vulnerabilities (V1) and (V2) can be jointly exploited to break ballot privacy for target voters: *i.e.,* voters the attacker decides to target and attack when they cast their ballot. We shall see that a channel attacker can learn how a target voter voted provided that there are at least 2 ballot-boxes in the target voter's constituency (this is the case for all the French constituencies). We present the attack core ideas in a simplified setting before presenting the actual attacks.

Attacks overview. Let us assume a channel attacker willing to learn the vote of Alice, who is registered in ballot-box $\mathtt{u}_1$ ($\mathtt{u}_1$ is a ballot-box identifier $\mathsf{ballotBoxId}$). The key ingredient for this attack is *inter-ballot-box move*: the attacker can cast Alice's ballot in $\mathtt{u}_2 \neq \mathtt{u}_1$ without anyone noticing. *One* approach, detailed in [Move$_H$] below, to achieve this is for the attacker to:

(i) move Alice's ballot b_A to u_2 and, thanks to (V1), give her a forged receipt corresponding to b_A as if it was cast in u_1.

(ii) to preserve the number of ballots in each ballot-box, the ballot b_B of a voter eligible in u_1, say Bob, can be moved to u_1 using the same technique.

Finally, because of (V1) Alice's and Bob's receipts pass individual checks, and because of (V2), the ballot-boxes $\mathtt{u}_1$ and $\mathtt{u}_2$ will be tallied without raising any error, despite the move of b_A to $\mathtt{u}_2$.

Now, let us assume the attacker knows a ballot-box $\mathtt{u}_2$ with a few eligible voters. Intuitively, the attack is as follows: the attacker exploits the inter-ballot-box ballot move attack to cast Alice's ballot b_A in the ballot-box $\mathtt{u}_2$. Exploiting the attack [Replace] the attacker replaces all other ballots cast in $\mathtt{u}_2$ by ballots $b_1, \ldots, b_n$ whose values are known. Because the ballot-boxes are individually tallied, the attacker will learn the result for $\mathtt{u}_2$ from which the Alice's vote encrypted in b_A can be deduced (since the votes encrypted in $b_1, \ldots, b_n$ are assumed to be known). We present below concrete attacks exploiting these ideas and explain how the attacker can completely evade detection. We then present fixes.

4.3.1 Attacking Ballot Privacy

As informally described above, our ballot privacy attacks rely on two ingredients: inter-ballot-box move, that is how to stealthily move a ballot from a ballot-box to another, and how to exploit the latter to introduce the privacy leak through tallying. We first present three inter-ballot-box ballot move techniques and then explain how any one of those can be exploited to mount ballot privacy attacks.

For each of those three inter-ballot-box move techniques, we assume that the election (constituency) identifier `elec` contains at least two ballot-boxes (consulates) identifiers $\mathtt{u}_1$ and $\mathtt{u}_2$. Alice is eligible to vote in consulate $\mathtt{u}_1$ and is the target voter whose vote will be moved from $\mathtt{u}_1$ to $\mathtt{u}_2$. To preserve the right number of ballots in each ballot-box, an arbitrary ballot (say Bob's) cast in $\mathtt{u}_2$ is moved to $\mathtt{u}_1$ (and possibly replaced).

[Move$_H$] Inter-ballot-box ballot move exploiting (V1,V2) for a voting server attacker. This first technique assumes a voting server attacker, *i.e.,* the voting server is compromised. The attack scenario is as follows:

Step a: Alice votes and sends a ballot b_A.

Step b: The voting server attacker receives b_A but stores it in the ballot-box $\mathtt{u}_2$, while Alice was eligible in $\mathtt{u}_1$.

Step c: In addition, the voting server attacker computes malicious receipts that will be valid for b_A, even though it was placed in the wrong $\mathtt{u}_2$:

$$H^{s_2} = H^{s_4} = m \| \mathsf{hash}(b_A \| m \| \mathtt{u}_2)$$
$$H^{s_1} = H^{s_3} = H^c = m \| \mathsf{hash}(b_A \| m \| \mathtt{u}_1)$$
$$\mathsf{cSU} = m' \| \mathtt{u}_1 \| \mathsf{hash}(b_A) \| \mathsf{sign}_{\mathsf{sk_S}}(\mathsf{hash}(m' \| \mathtt{u}_2 \| \mathsf{hash}(b_A)))$$

where $m' = \mathsf{roundId} \| \mathsf{electionId} \| \mathsf{electionName}$ and $m = \mathsf{roundId} \| \mathsf{electionId}$ are election meta-data. Note that H^c, H^{s_1}, and H^{s_3} are computed as expected but H^{s_2} and H^{s_4} are modified such that the receipt is valid for b_A cast in $\mathtt{u}_2$. This is crucial to make all 3$^{\text{rd}}$-party checks succeed and avoid detection (we come back to this in Section 4.3.2).

Step d: Steps a, b, and c are applied to move a ballot b_B of an arbitrary voter Bob eligible in $\mathtt{u}_2$ from $\mathtt{u}_2$ to $\mathtt{u}_1$.

[Move$_b$] Inter-ballot-box ballot move exploiting (V1,V2) for a channel attacker. In this scenario, we assume that the attacker only controls the (plaintext) communication channel.

The attack scenario is as follows:

Step a: Alice votes and sends a ballot b_A.

Step b: The channel attacker intercepts the ballot b_A and replaces it by a new ballot b_{att} they choose. In addition, exploiting (V1) (attack [Replace]), the attacker modifies the messages sent by the voting server to make Alice get a valid receipt for b_{att}, while she intended to cast b_A instead, *i.e.,*

$$H_1^{s_2} = H_1^{s_4} = m \| \mathsf{hash}(b_{att} \| m \| \mathtt{u}_1)$$
$$H_1^{s_1} = H_1^{s_3} = H_1^c = m \| \mathsf{hash}(b_A \| m \| \mathtt{u}_1)$$
$$\mathsf{cSU}_1 = m' \| \mathtt{u}_1 \| \mathsf{hash}(b_{att}) \| \mathsf{sign}_{\mathsf{sk_S}}(\mathsf{hash}(m' \| \mathtt{u}_1 \| \mathsf{hash}(b_{att})))$$

Note that H^c, H^{s_1}, H^{s_3} are computed as expected but H^{s_2} and H^{s_4} are modified such that they will be valid against a ballot-box $\mathtt{u}_1$ containing b_{att} instead of b_A. In particular, cSU_1 is exactly the seal returned by the voting server when it received b_{att}; the attacker does not need to forge a signature.

Step c: Similarly, the attacker intercepts b_B a ballot sent by Bob, an eligible voter in $\mathtt{u}_2$, and replaces it with Alice's ballot b_A exploiting (V1) (attack [Replace]). Again the attacker modifies the references H_2 and the seal cSU_2 sent to Bob replacing b_B by b_A.

[Move$_{ID}$] Inter-ballot-box ballot move without exploiting (V1) for a voting server attacker. We assume a server attacker that does not exploit (V1). Because ballotBoxId is sent by the voting server to the voting client, the server can send $\mathtt{u}_2$ instead of $\mathtt{u}_1$ to make the Alice's voting client compute a ballot for $\mathtt{u}_2$ by itself. The displayed data and the individual checks do not allow Alice to detect that the ballot was computed for the wrong ballotBoxId. Again, to preserve the number of ballots in each ballot-box, we assume that the voting server sends $\mathtt{u}_1$ instead of $\mathtt{u}_2$ to Bob, an arbitrary eligible voter in $\mathtt{u}_2$ so that Bob computes and casts a ballot for $\mathtt{u}_1$.

[Swap$_X$] Exploiting inter-ballot-box ballot move to attack ballot privacy. We now explain how any of the three move techniques can be exploited to violate ballot privacy with three attack variants: [Swap$_H$], [Swap$_b$], and [Swap$_{ID}$]. In the following, we specify the attack [Swap$_X$] for $X \in \{H, b, ID\}$.

We already described in the *attacks overview* paragraph (section 4.3) the main idea of gathering Alice's ballot with ballots whose votes are known to the attacker in a single ballot-box u_2 so that when tallied, it reveals Alice's vote. In order to make this completely stealthily, the attacker needs to take some precautions.

First, the attacker needs to preserve a perfect correspondence between the number of ballots in each ballot-box and the number of voters eligible in this ballot-box who did cast a ballot. Indeed, any voter who casts a vote is registered in a semi-private signing sheet. We explain in Appendix B, Remark 11 that it is unlikely that voters check the signing sheet due to various practical constraints. Even if they did, the attack [Move$_X$] takes care of preserving the number of ballots in each ballot-box, and thus prevents any potential detection.

Second, the attacker must also make sure that the moves and replacements he may operate will not make a voting option that was intended to be expressed in ballot-box $\mathtt{u}_1$ or $\mathtt{u}_2$ completely disappear, as already explained in Section 4.2. This is addressed with Step b and Step c below. Note that we assume that at least $|\mathcal{V}|$ voters eligible in $\mathtt{u}_2$ (resp. in $\mathtt{u}_1$) are willing to cast a ballot, which is a reasonable assumption (see Section 4.3.2).

The attack [Swap$_X$] is as follows. For the target Alice eligible in $\mathtt{u}_1$, the attacker first selects a ballot-box $\mathtt{u}_2$, ideally with a low number of voters and does the following:

Step a: The attacker uses [Move$_X$] to move Alice's ballot b_A from $\mathtt{u}_1$ to $\mathtt{u}_2$ and some arbitrary Bob's ballot b_B from $\mathtt{u}_2$ to $\mathtt{u}_1$ (with a replacement of b_B by b_{att} if $X = B$).

Step b: For any other voter willing to cast a ballot b in $\mathtt{u}_2$, the attacker exploits [Replace] to replace b by an attacker-chosen ballot b'. The attacker can choose b' such that: (i) each voting option $v \in \mathcal{V}$ gets at least one vote in $\mathtt{u}_2$ and (ii) the overall vote distribution is close to the expected one (to not raise any suspicion).

Step c (optional): If the expected vote distribution in $\mathtt{u}_1$ makes it likely that a voting option $v \in \mathcal{V}$ might get no vote, then [Replace] must also be used against $|\mathcal{V}|$ voters eligible in $\mathtt{u}_1$ to remedy this problem. This would not have been required for most ballot-boxes in the 2022 election (see Remark 6).

Finally, the tally of $\mathtt{u}_2$ leaks Alice's intended vote.

Remark 6. *As presented above, the attack [Swap$_{ID}$] relies on (V1) at Step b and Step c. This is actually not necessary with the following additional assumptions about other voters (where $\mathcal{V} = \{v_1, \ldots, v_k\}$):*

1. *There are $E \geq k+1$ eligible voters in $\mathtt{u}_2$.*
2. *The attacker* knows *k voters, $V_1, \ldots, V_k$, that are different from Alice and such that V_i is willing to vote for v_i (in $\mathtt{u}_1$ or $\mathtt{u}_2$). Those voters are either colluding with the attacker or are honest but the attacker has a good guess about how they will vote.*
3. *There* exist *k other voters, $V_{k+1}, \ldots, V_{2k}$, different from Alice such that V_{k+i} is willing to vote for v_i in $\mathtt{u}_1$ for all $i \in \{1, \ldots k\}$. In contrary to $V_1, \ldots, V_k$, we do not assume that the attacker knows $V_{k+1}, \ldots, V_{2k}$, we solely assume that they exist. In practice, the attacker can just be convinced they exist based on statistical data (e.g., previous election results). For example, almost all of the ballot-boxes from the 2022 election got at least one vote for each of the candidates.*
4. *The attacker* knows *$E-1$ voters $V'_1, \ldots, V'_{E-1}$ eligible in any consulate (of the same election as Alice's) and how they are willing to vote. Those voters are either colluding with the attacker or are honest but the attacker has a good guess about how they will vote.*[11]

Those assumptions are reasonable in practice due to the existence, for each constituency, of both a ballot-box $\mathtt{u}_2$ with very few eligible voters (dozens) and a ballot-box $\mathtt{u}_1$ with a large number of voters (thousands).

How to achieve Step b and Step c of [Swap$_{ID}$] without relying on (V1) is slightly technical and detailed in Appendix B.2. [Swap$_{ID}$] thus breaks ballot privacy for target voters, assuming a server attacker and requires dedicated fixes as it does not rely on (V1).

4.3.2 Impact and Stealthiness

The impact of our privacy attacks is maximal when all plaintext votes of the ballots in the resulting ballot-box $\mathtt{u}_2$ are known to the attacker, except for the one of the target voter (*e.g.,* Alice).[12] In this case, the result of the tally of $\mathtt{u}_2$ directly reveals Alice's vote to the attacker. The larger is $\mathtt{u}_2$, the more replacements are needed. We now discuss how suitable $\mathtt{u}_2$ can be chosen using data of the election for which FLEP was used [4] and then discuss coercion and stealthiness.

Many suitable $\mathtt{u}_2$. In the 2022 French legislative election, many consulates received a small number of votes, hence suitable choices for $\mathtt{u}_2$. For example MSQ-MINSK during the second round (6 electronic votes among 129 eligible voters) or PBM-PARAMARIBO during the second round (5 electronic votes among 145 eligible voters). These two consulates belong to constituencies having consulates with a large number of voters with potential targets, hence suitable choices for $\mathtt{u}_1$, such as SYD-SYDNEY which received 4049 electronic ballots during the second round, or MEX-MEXICO with 1959. There were even extreme cases with some consulates which received a *unique* ballot as discussed in Appendix B.2. Obviously, our attacks can be exploited against different targets voters (Alice) using different target ballot-boxes ($\mathtt{u}_2$).

New coercion power. Moreover, the attacker can also target several voters instead of just Alice and gather all their ballots into $\mathtt{u}_2$. This attack would give a power of coercion, in a remote way[13], against those voters: the tally of $\mathtt{u}_2$ will reveal the vote distribution among them and thus make the coercer able to detect that some decided to not comply with the attacker's instructions and how many (the attacker does not learn their identity though). Similarly, the attacker can exploit this to learn the vote distribution of a population of voters of special interest, *e.g.,* industrial actors. Those votes would normally be mixed with many other votes in their respective ballot-boxes, thus protecting their privacy.

Why are our privacy attacks completely stealthily? We review all checks that are performed by the different actors and explain in details why they fail to detect our privacy attacks in Appendix B.2. We summarize this study here.

[11] Strictly speaking, if those voters are not colluding and are eligible in a consulate $\mathtt{u}$ different from $\mathtt{u}_1$ and $\mathtt{u}_2$, then a similar assumption as 3. should be made about $\mathtt{u}$, that is it is expected that all voting options will be voted for, excluding those V'_i.

[12] Note that, even if some unknown ballots remain in $\mathtt{u}_2$, some privacy leak still exists. Some quantitative analyses of similar privacy leaks have been conducted [28] and have revealed that it could be harmful.

[13] Note that the FLEP was not intended to guarantee coercion-resistance, which is usually understood as a resistance to over-the-shoulder coercion where the attacker is physically with the voter under coercion. However, one could have reasonably expected that it was resistant to this weak form of *remote* coercion, in which the coercer has *never* access to the voters' devices.

From the (honest) voting server point of view, we note that data received and processed by the voting server are genuine- and honest-looking, the voting server does not know how voters intended to vote. From the voting client point of view, the received malicious ballot receipts pass all consistency checks and no error can be detected. No cheating can be detected from the ballot-boxes tallied result either since each ballot-box contains (at least) one ballot for each voting option. Therefore, even if a voter's ballot has been moved in another ballot-box or replaced, then they will always see in the result at least one vote which corresponds to their intent, it could have come from their ballot.[14]

We now discuss public information and independent verification services proposed to the voters. The 3rd-party checks that the tally of the ballot-box has been correctly computed and fails to detect any cheating because of (V2). The voters' individual verifiability checks also fail to detect our attacks, even when carried out with the 3rd-party service. Indeed, to answer these queries, the latter recomputes all the references H and hashed values hb of the ballots provided by the authorities, which are then looked up. It is important to note that, because the 3rd-party was aiming to guarantee verifiability, this look-up is performed taking all the ballots of the given constituency identifier `elec` into account. Therefore, even malicious receipts will be found and deemed valid.

Finally, the human voters themselves are unable to detect that the ballot reference they were shown ($H_1^{s_2} = H_1^{s_4}$) are not computed with the right ballotBoxId (and the genuine, honest ballot b). Indeed, to be able to detect this, the voters would need to recompute H and thus to know what is hashed in H (*e.g.,* b) but voters are never shown b.

Impact: A channel or a voting server attacker has the technical ability to learn some target voters' vote without breaking any encryption. This attack would leave no evidence we could then exploit to know if the attack happened.

4.3.3 Fixing Ballot Privacy

We propose three solutions to prevent our attacks.

Fix 3: A first fix is to add the ballot-box (consulate) identifier ballotBoxId (*i.e.,* `u`) in the context of the ZKP. This simple modification cryptographically prevents any swap of ballots between different ballot-boxes once it has been computed.

Fix 4: Even if Fix 3 was implemented, [swapID] is still possible. Therefore, we recommend coupling Fix 3 with displaying in the voting client the ballotBoxId used for computing the ballot (or the readable name of the corresponding consulate). This way, the voter could notice the cheating prior to casting.

Fix 5: In addition to Fix 3, the 3rd-party verification service may verify the consistency between the ballotBoxId in the ZKP (and the cSU) and the ballot-box in which it is stored as introduced in the tool after the elections run in 2022 [13]. Therefore, as an alternative to Fix 4, the 3rd-party could additionally display to the voter the consulate in which their ballot has been cast and counted.[15] This solution is easy to implement but only prevents privacy attacks against voters who use the 3rd-party verification service. This is of broader interest: 3rd-party, *i.e.,* an external auditor, can be useful not only to ensure verifiability, but also vote privacy.

Remark 7. *Fix 3 and 5 have been implemented by Voxaly Docaposte and 3rd-party thanks to our findings and were used for elections in 2023.*

Fixing the vulnerability (V1) with Fix 1 or Fix 2 would already prevent all but the [$swap_{ID}$] attack. The latter requires the combination of Fix 3 and either Fix 4 or Fix 5. Moreover, we believe the second vulnerability we found (V2) should be addressed independently with Fix 3 as it introduces a weakness, even though not exploitable on its own. Therefore Fix 3 and 4 would be the best option.

5 Lessons Learned

Our reverse and security analysis of the FLEP revealed serious vulnerabilities and attacks in the protocol and its deployment. This is quite impactful on its own given the importance (nation-wide legislative election) and scale (largest political election using e-voting) of this election. Our analysis of the FLEP is also relevant for its illustration of how things can go bad with the excessively challenging real-world deployment at scale of e-voting solutions. Some of them are due to pitfalls that we point out for improving future deployments of e-voting in general. Others are due to limitations and problems with state-of-the art academic e-voting solutions from which we derive practically relevant open research questions of general interest. We summarize those lessons that go beyond the specific case of the FLEP and we refer the curious reader to Appendix E for more in-depth discussions.

[14] The only assumption we make here is that honest voters do not collude to infer what should be the expected, honest result. Note that, the attacker can remain stealthily even under such extreme circumstances: he could rely on malicious and colluding voters in the same ballot-box to lie and cheat against the honest voters.

[15] Note that ballotBoxId is also in plaintext in cSU so it could be directly checked but it seems unrealistic to ask human voters to do so.

Voting client as critical component. Great care has been put in securing the FLEP voting server against internal and external threats, which is a common practice. E-voting requires to adopt a radically different mindset in that regard as it relies on *verifiability* so that no one has to trust the voting server, its administrators, the code it runs, etc. (sometimes called *software independence* [32]). The most important component of the protocol that should be the main target of audits and analyses is actually the *voting client* and the verification services (*e.g.,* 3rd-party).[16] If the voting client is securely designed and implemented, the protocol should guarantee the election integrity independently of the voting server security. This is well illustrated by our work and our attacks: the partial specification and 3rd-party verification services are totally voided by the voting client being flawed. An implementation weakness in the voting client can completely defeat verifiability and privacy as we have shown. *Recommendation:* Consider the voting client and the verification services as the main targets of analyses and internal audits. We advocate for making them amenable to public scrutiny with a comprehensive public specification and open-sourced code.

The FLEP, as well as many e-voting systems assume an honest voting device (*e.g.,* [10, 15, 26]). On the surface, this seems unrealistic since the latter is distributed by a possibly compromised server or channel. Instead, it could be distributed through an app marketplace to benefit from code integrity (assuming the marketplace is trustworthy). Otherwise, when downloaded and run in the browser, it is inherently vulnerable to integrity flaws. This is resolved in the literature with the notion of "auditability" (*e.g.,* [7])[17]: *external auditors* (who can be any voter or external actors) can pretend to be voters to compare the code they receive with the legitimate version and thus implement some form of "universal integrity checks". We stress that such checks are by no means an audit of the code (that is rather the role of the commissioned auditors) but rather a mere comparison of two code bases – or the hashes thereof. If there are enough auditors well hidden among the voters, the voting server takes high risk of being caught when tampering with the voting client. In the context of the FLEP, such external audits were never specified and were made complex to carry over[18]. This can be addressed by using Single-page Application (SPA)[19] to distribute the voting client, as is the case for Belenios. Only the SPA needs to be compared to the legitimate version and can be considered as a static file, which can then be considered to be honest (assuming the legitimate version has been well audited by commissioned auditors, once-for-all). *Recommendation:* Distribute the voting client with a standalone app or an SPA and clearly specify external auditors' tasks, notably universal integrity checks.

Finally, the vulnerability (V1), which is essentially a verifiability weakness, is key to our privacy attacks. Such a verifiability-privacy relation has been documented before with a theoretical attack [17], our work illustrates further this relation with practical attacks. *Lesson:* Privacy can be attacked in practice due to a lack of verifiability.

Operational constraints as scientific bottleneck. For deploying a protocol in the real world and especially for political elections, many aspects that are very often omitted in the academia have to be taken into account such as the legal requirements and recommendations, compliance to a competitive public call for tenders, etc. This partly explains why, despite being derived from the proven secure Belenios [15], the FLEP ended up flawed with weaknesses that were inexistent in Belenios.

Missing features. First, the FLEP had to accommodate the multi-ballot-box setting. This setting seems to be a recurring pattern. Importantly, it is the source of quite impactful privacy attacks: [Swap_b,Swap_H,Swap_{ID}] for the FLEP and another privacy attack [11] for the Swiss Post 2022 protocol, which also relies on multiple ballot-boxes but exploits a different weakness. Another lacking feature in Belenios was the downloadable receipt, a desired feature of the EFA Ministry from their interpretation of the CNIL recommendations. As a result, the PDF receipt has been introduced in the FLEP but was not properly checked. *Research question:* Make state-of-the-art solutions more generic so that they can accommodate such real-world use cases and practical constraints. Moreover, formal security properties and proofs do not consider such features and could thus miss practical attacks.

Distribution of authentication. The FLEP suffers from a lack of trust distribution for the voter authentication process: a single entity, the voting server, is in charge of this authentication as it stores and checks the two authentication factors. A compromised voting server can thus impersonate voters and do *ballot stuffing*. A simple theoretical solution to fix this weakness is to distribute the generation and the verification of the authentication factors. Unfortunately, deploying such an infrastructure is difficult. For example, if the independent 3rd-party entity had to participate to user authentication, then it would have to offer an API for authentication verification. *Research question:* As far as we know, there is no practical academic solution to this problem that matches real-world constraints of deployments. We envision the spread of eID cards or the development of standards like openID connect [33] could be built upon for this.

Secret Key Generation distribution. We show in Appendix D.3 that some lawful requirements governing when a quorum is met to *e.g.,* decrypt a ballot-box are not cryptographically enforced in the FLEP. (The lawful requirements combine conditions such as: a holder and a substitute for a given role cannot be simultaneously in the same quorum, a quorum should have at

[16] This does not dispense to make effort to secure the voting server.

[17] Note that mechanisms such as the Belanoh challenge [8] do not ensure the voting client integrity.

[18] The JavaScript code is served post-authentication, the voting client logic is spread all over HTML and JavaScript files, interleaved with user- and session-specific data (not easily comparable)

[19] https://developer.mozilla.org/fr/docs/Glossary/SPA

least 4 roles represented, etc.) As a consequence, fewer people than what is legally prescribed by the law can collude and decrypt a ballot-box. State-of-the-art academic solutions cannot be used off-the-shelf to address this. *Open Questions:* Is it possible to cryptographically enforce such operational constraints to prevent any human misbehavior? Are there solutions that are generic enough to be suitable for practically relevant constraints seen in real-world use cases? We formalize such questions in Appendix E.2.1.

Clear security objectives and threat model. Neither the FLEP nor legal requirements and recommendations clearly define the expected security objectives and threat models. It is meaningless to assess the security of a system without a clear threat model definition, that is why we first had to do it ourselves. It is also of little interest for the public to know that the FLEP has undergone audits without having access to the scope and the objectives of this audit. *Recommendation:* Election organizers, or even better the responsible for the public call for tenders (EFA Ministry), should clarify and specify their expectations regarding the security objectives and threat model. This will certainly help academic researchers to identify and address practically relevant problems, incentivize more secure solutions in calls for tenders, and allow more relevant audits and security analyses. Ideally, objectives and threat models could be prescribed by law, as it is the case in Switzerland (see Remark 16), where even mathematical cryptographic and symbolic proofs are required. We also recommend requiring such proofs.

Simpler and precise voting journey instructions. In the FLEP, the voters receive different confusing and quite overwhelming information about the verification process. In total, they are shown 4 quite intimidating cryptographic data and are offered 4 different verification tasks, see Appendix E.3.1. *Recommendation:* Complex protocol and notably voter's journey and verification tasks are detrimental to the overall protocol security. For the FLEP, we explain in Appendix E.3.1, how the protocol can be conservatively simplified with only one cryptographic datum shown to the voters and a unique verification task, which would certainly have avoided the vulnerability (V1). In general, we recommend prioritizing any simplification impacting the voters' journey and clearly specifying what is expected from the voters and assume no more from them.

Transparency and openness. The FLEP was lacking a clear (and open) specification, which partially explains why its flaws remained unnoticed until we saw them, while the election was running. (We present in Appendix D additional weaknesses that are well-known in the literature, *e.g.,* ballot replay attacks, and that nonetheless also affect the FLEP.) *Recommendation:* Because designing and deploying e-voting solutions is a notoriously difficult task, we recommend promoting transparency and public scrutiny from different communities (academic researchers, hackers, etc.) to detect and prevent vulnerabilities as early as possible. This could be incentivized with public intrusion test and bug bounty programs, as done in Switzerland where open access is even a legal requirement.

6 Conclusion

The FLEP protocol is an instructive example of a real world deployment of a variant of an academic protocol (Belenios) that introduces design-level and implementation-level weaknesses. The latter opened up the 1.6 million eligible voters overseas of the 2022 French legislative election to integrity and privacy attacks under a voting server attacker or a weaker channel attacker. To avoid such failures in the future, we proposed fixes, which have been (almost fully) implemented, and have made various recommendations we learned from this experience. We had insightful discussions with the different stakeholders and we strongly believe more of such communication between academia and e-voting practitioners is for the better.

However, we must remain cautious as, in the absence of formal security proofs, it is still possible that other critical attacks might affect the 2023 version of the FLEP protocol. Narrowing down to the weaknesses we know of, we recall that better solutions should be proposed for authenticating the voters, getting rid of the strong assumption of a trustworthy voting client, etc. Indeed, in its current state, the FLEP protocol still allows a compromised voting server to stuff the ballot-box by impersonating voters who do not vote. Such an attacker can also modify the voters' intended vote having a corrupted voting device, *e.g.,* because of a malicious web browser extension. Moreover, unlike the flaws we discovered, we consider that there is currently no off-the-shelf solution to prevent such attacks. This might be considered as a warning that the state of the art in e-voting is not ready for such critical elections and an incentive to continue the research towards practical solutions to those problems.

Responsible Disclosure and Acknowledgments

We conducted this security analysis during the 2022 election period through a passive analysis only; we never attacked voting servers. Therefore, we could not alter the integrity or the security of the election. All the vulnerabilities reported in this document have been reported to the relevant stakeholders right after the election and at least 3 months before publication. We thank those stakeholders, *i.e.,* EFA Ministry, ANSSI, Voxaly Docaposte, and the researchers running the 3rd-party services (Stéphane Glondu, Pierrick Gaudry, and Véronique Cortier) for their help and discussions after we sent them our findings.

In particular, we would like to thank again the role of ANSSI in the responsible disclosure process, which has always be a key player in promoting transparency and openness. This is greatly appreciated given the context of this work.

Finally, we would like to thank our colleagues Myrto Arapinis, Hugo Labrande, and Emmanuel Thomé for their help to collect data about the FLEP.

Conflicts of interest statement

The researchers running the 3rd-party services (Stéphane Glondu, Pierrick Gaudry, and Véronique Cortier) are colleagues of us. However, they were under embargo when we did our own research and were forbidden to communicate with us on that matter. Our research was carried out in a completely independent way.

References

[1] Results of the first round of the French Legislative elections 2022: https://www.diplomatie.gouv.fr/fr/services- aux-francais/voter-a-l-etranger/resultats-des-elections/article/elections- legislatives-resultats-du-1er-tour-pour-les-francais-de-l-etranger.

[2] Large scale test of the FLEP: https://amsterdam.consulfrance.org/ Elections-legislatives-2022-vote-par-internet-second-test-grandeur-nature.

[3] Firefox sessionStorage documentation: https://developer.mozilla.org/fr/docs/Web/API/Window/sessionStorage.

[4] Results of the second round of the French Legislative elections 2022: https://www.diplomatie.gouv.fr/fr/services- aux-francais/voter-a-l-etranger/resultats-des-elections/article/elections- legislatives-resultats-du-2eme-tour-pour-les-francais-de-l-etranger.

[5] Ben Adida. Helios: Web-based open-audit voting. In *17th conference on Security Symposium (SS'08)*. USENIX Association, 2008.

[6] Christof Beierle, Patrick Derbez, Gregor Leander, Gaëtan Leurent, Håvard Raddum, Yann Rotella, David Rupprecht, and Lukas Stennes. Cryptanalysis of the gprs encryption algorithms gea-1 and gea-2. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 155–183. Springer, 2021.

[7] 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 *Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE'13)*, 2013.

[8] Josh Daniel Cohen Benaloh. *Verifiable secret-ballot elections*. Yale University, 1987.

[9] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. Sok: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy*, pages 499–516. IEEE, 2015.

[10] Michael R Clarkson, Stephen Chong, and Andrew C Myers. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy (SP'08)*, pages 354–368. IEEE, 2008.

[11] Véronique Cortier, Alexandre Debant, and Pierrick Gaudry. A privacy attack on the Swiss Post e-voting system. Research report, Université de Lorraine, CNRS, Inria, LORIA, November 2021. https://hal.inria.fr/hal-03446801/file/rwc.pdf.

[12] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. Informal description of the Vvfe web services. In 2022: https://verifiabilite-legislatives2022.fr/. In 2023: https://verifiabilite-legislatives2023.fr/.

[13] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. `wip` branch (commit 5251b2f2) of the Vvfe tool. https://gitlab.inria.fr/vvfe/vvfe.

[14] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. Vvfe: Vérifiabilité du vote des français de l'étranger. https://gitlab.inria.fr/vvfe/vvfe.

[15] Véronique Cortier, Pierrick Gaudry, and Stephane Glondu. Belenios: a simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning*, pages 214–238. Springer, 2019.

[16] Véronique Cortier, Pierrick Gaudry, and Quentin Yang. How to fake zero-knowledge proofs, again. In *E-Vote-Id 2020 - The International Conference for Electronic Voting*, Bregenz (virtual), Austria, 2020.

[17] Véronique Cortier and Joseph Lallemand. Voting: You can't have privacy without individual verifiability. In *ACM SIGSAC conference on computer and communications security (CCS'18)*, pages 53–66, 2018.

[18] Veronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. In *2011 IEEE 24th Computer Security Foundations Symposium*, pages 297–311, 2011.

[19] Voxaly Docaposte. Partial specification of the flep, 2022. Available at the link https://w8t9w2j6.stackpathcdn.com/wp-content/uploads/VOXALY_LEG2022_Verifiabilite_Specifications.pdf obtained from https://www.voxaly.com/vote-par-internet-pour-les-francais-de-letranger-dans-le-cadre-des-elections-legislatives-2022.

[20] Voxaly Docaposte. Partial specification of the flep, 2023. Available at the link https://www.voxaly.com/wp-content/uploads/VOXALY-LEG2023-Transparence-et-Verifiabilite-Specifications-publiques-v2-04.pdf obtained from https://www.voxaly.com/vote-par-internet-pour-les-francais-de-letranger-dans-le-cadre-des-elections-legislatives-partielles-2023.

[21] Code électoral. French law governing political elections. https://www.legifrance.gouv.fr/codes/section_lc/LEGITEXT000006070239/LEGISCTA000006115482.

[22] Pierrick Gaudry and Alexander Golovnev. Breaking the encryption scheme of the moscow internet voting system. In *Financial Cryptography and Data Security (FC'20)*, pages 32–49. Springer, 2020.

[23] Stéphane Glondu. Belenios specification - version 1.17. http://www.belenios.org/specification.pdf, 2021.

[24] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *IEEE Symposium on Security and Privacy (SP'20)*, pages 644–660. IEEE, 2020.

[25] Sven Heiberg, Tarvi Martens, Priit Vinkel, and Jan Willemson. Improving the verifiability of the estonian internet voting scheme. In *Electronic Voting (E-Vote-ID'16)*, pages 92–107. Springer, 2017.

[26] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *ACM Workshop on Privacy in the Electronic Society*, pages 61–70, 2005.

[27] Ralf Kusters, Tomasz Truderung, and Andreas Vogt. Clash attacks on the verifiability of e-voting systems. In *IEEE Symposium on Security and Privacy (SP'12)*. IEEE, 2012.

[28] David Mestel, Johannes Müller, and Pascal Reisert. How efficient are replay attacks against vote privacy? a formal quantitative analysis. In *IEEE 35th Computer Security Foundations Symposium (CSF'22)*, pages 179–194. IEEE, 2022.

[29] Johannes Mueller. Breaking and fixing vote privacy of the estonian e-voting protocol ivxv. In *Workshop on Advances in Secure Electronic Voting*, 2022.

[30] Journal Officiel. Délibération n° 2019-053 du 25 avril 2019, 2019. Available at the link https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000038661239.

[31] Léo Perrin and Xavier Bonnetain. Russian style (lack of) randomness. In *Symposium sur la sécurité des technologies de l'information et des communications*, 2019.

[32] 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), 2008.

[33] Natsuhiko Sakimura, John Bradley, Mike Jones, Breno De Medeiros, and Chuck Mortimore. Openid connect core 1.0. *The OpenID Foundation*, 2014. https://openid.net/specs/openid-connect-core-1_0.html.

[34] Swiss Post. e-voting system. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation.

A Security Objectives and Threat Model

We continue Section 2.2 with more details about how we derived the security goals and threat model of the FLEP.

In France, e-voting is prohibited for almost all political elections. Elections organized for citizens overseas (*i.e.,* legislative and consular elections for consulates outside France) are the unique exceptions. As an immediate consequence, it results in a lack of a precise description of the security goals, trust assumptions, and threat models that should be considered. Fortunately, there exist requirements that the FLEP should comply with, notably the Code électoral [21] (*i.e.,* the French law governing elections) which defines the lawful requirements that apply to e-voting during the legislative election, and the recommendations enacted by the CNIL [30]. As mentioned above, the FLEP is supposed to achieve the highest level of security defined by the CNIL, i.e., level 3.

Remark 8. *Even if fulfilling the CNIL recommendations is not, strictly speaking, a legal requirement, they explicitly define the expected security of all the e-voting systems in use in France. Given the critical nature of the French legislative election, it would be unreasonable for the FLEP to not follow all of those recommendations. The first level corresponds to small elections with a small number of voters, a small impact, and a low risk of attacks or compromise. These elections correspond for instance to representative election in small sports associations. On the contrary, the third level, the highest, has been defined for elections with a large number of voters, a large impact, and a high risk of attack attempts or compromises. This matches all the characteristics of a political election.*

We quote some of those requirements to consolidate a set of trust assumptions and threat models associated with the key security goals of e-voting protocols in the literature: verifiability and ballot privacy (also called vote secrecy). Those quotes (in gray boxes) are translated by us. (We provide the original version with their translations in Appendix F.)

A.1 Ballot Privacy

First, the Code électoral and the CNIL requirements agree on the fact that the FLEP must ensure the *confidentiality of the votes*.

> "Votes must remain confidential"
>
> —Code électoral, Article R176-3-9 [21]

> "[the system must] ensure the strict confidentiality of the ballots as soon as created."
>
> —CNIL, Security objective n° 1-04 [30]

> "[The system must] ensure that the identity of the voter and the expression of his choice can not be linked during the whole process"
>
> —CNIL, Security objective n° 1-07 [30]

Because the notion of confidentiality of the votes may be subject to different interpretations depending on the context (academic papers, law, public discussions...), we provide a (still informal) definition to clarify the notion we shall refer to in this paper.

Definition 1 (Confidentiality)**.** *An e-voting protocol ensures* confidentiality of the votes *if an attacker is unable to learn the (plaintext) vote of a target voter.*

Remark 9. *The security property of Definition 1 is strictly weaker than state-of-the-art academic definitions such as the* ballot privacy *definitions of Benaloh* et. al. *[8] or Bernhard* et. al. *[9]. The latter considers that an attacker should not be able to learn any bias about how the target voter voted for.*

Since we shall show that the FLEP does not satisfy even the weak notion of confidentiality *from Definition 1 for target voters under a channel attacker, we do not use a formal definition of* vote secrecy *or* ballot privacy *in this document.*

A.2 Verifiability

Second, the FLEP must ensure the integrity of the election outcome. The academic literature usually splits this into three sub-properties:

- **eligibility:** all the ballots that are counted during the tally have been cast by legitimate voters;

- **universal verifiability:** the result of the election corresponds to the content of the ballot-box that has been tallied;
- **individual verifiability:** each voter is able to verify that their ballot has been added into the ballot-box.

In this document we focused on the last property, *individual verifiability*, that we showed the FLEP fails to guarantee. Individual verifiability immediately relates to the Code électoral and the CNIL recommendations as follows:[20]

> When a voter's vote is registered, the voter is provided with a digital receipt allowing them to verify online that their vote has been taken into account.
>
> —Code électoral, Article R176-3-9 [21]

> ensure the transparency of the ballot-box for all the voters [...] It must be possible for the voters to ensure that their ballot has been counted in the ballot-box.
>
> —CNIL, Security objective n° 2-07 [30]

As we shall see, the FLEP uses some hashed values over the cast ballot and some meta-data as well as a signature as digital receipt. The protocol offers *verification* services, where voters enter their receipt and get notified whether their ballot has been taken into account.

To do so, the FLEP does not rely on a public bulletin board to let the voter make these verifications. Hence, the voter must use services proposed by the voting server itself. In order to meet the highest level of security enacted by the CNIL (level 3, see below), the FLEP also relies on the 3rd-party role to provide these verification services in addition to the voting server. This relates to the security objective n°3-02 [30].

> The system must allow transparency of the ballot-box for all voters from third-party tools.
>
> —CNIL, Security objective n° 3-02 [30]

A.3 Threat Model

The legal requirements and CNIL recommendations do not provide a formal threat model to conduct our security analysis. Conversely, we are not lawyers, so we will never conclude that FLEP is not compliant to lawful regulations and requirements such as the CNIL objectives. Instead, we shall define a threat model that we argue is in line with the legal framework. Additionally, the threat model we shall define is the de-facto standard for most e-voting protocols from the academic literature.

Should the FLEP remain secure under a compromised voting server? Recall that due to its usage for large-scale, political, national election, FLEP must at least fulfill the CNIL's objectives under the CNIL Security Level 3 defined as follows.

> Security level 3: The threat actors include the voters, the election operators, outsiders, insiders within the provider or internal staff. They can be resourceful or highly motivated.
>
> —CNIL, Security level 3 [30]

The fact that internal threats are considered (election operators) and that one of the objectives for this level (Objective n°3-02) is to set up an independent third party server for verification purpose indicates that the FLEP should remain secure under at least a partial server compromise. A trustworthy voting server would make third-party verification services completely useless and spurious. To support this further, we also argue that:

- the FLEP is designed to provide verifiability and is presented as such. The purpose of verifiability is precisely to free the system from the trust assumption of a trustworthy voting server: with verifiability, the election outcome should be trustworthy even when the voting server is entirely compromised, as it is the case with many state-of-the-art protocols (*e.g.,* Helios, Belenios, etc.).
- the voting server is online and subject to external attacks. Even if many operational safeguards are implemented, making such a system uncompromisable even for resourceful or highly motivated internal or external threats, is an extremely difficult task.

[20] Note that the CNIL has also some recommendations for the other sub-properties.

- the voting server is operated by the EFA Ministry which is not necessarily representative of the population. Having to entirely trust the EFA Ministry and its officials would boil down to a radical trust shift, compared to paper-based voting, which is the main voting method replaced by e-voting. Indeed, in-person voting was representing 92,5% of expressed votes in the first round in 2017[21] (last French legislative elections where two options were offered to voters: in-person voting and voting by mail), while, in 2022, in-person voting was only used by 22,7% and e-voting with FLEP was used by 76% of voters [1].

- the voting server is running software provided by the vendor, which is a (private law) company that may have interests in political elections. Different audit mechanisms have been used and pen-testing campaigns have been conducted to try to prevent such a corruption but trapdoors are hard to detect in practice [6, 31]. Again, having to trust this vendor would represent a massive trust shift for such political elections.

Threat Model for the FLEP. In this paper, the main threat model we consider is actually even weaker than a compromised voting server, that is it considers an even weaker attacker. Indeed, instead of considering a (possibly) corrupted voting server, we shall assume a corrupted communication plaintext channel between the voters and the server. We call such an attacker a *channel attacker*. When referring to an attacker who can compromise a voting server, we shall explicitly write *voting server attacker*.

More precisely, the *channel attacker* can intercept and inject (plaintext) messages in-between voters under attack and the voting server but has not necessarily access to the voting server internals, such as its databases, the signing sheet, the authentication material, the log files, etc. Obviously a voting server attacker is also a channel attacker, but the converse is wrong in general.

Another way to realize a channel attacker is for example for the attacker to compromise the TLS certificate of the voting server in order to later act as a Mallory-in-the-Middle (MiM).[22] (Indeed, a channel attacker should have access to the plaintext of the exchanged messages.) Recall that the CNIL Security level 3 considers inside threats, in particular malicious election operators, so a TLS credential theft is in scope (even more so due to the use of 4 different servers for load-balancing and the use of middleboxes for filtering the traffic). Yet another way to realize a channel attacker is to exploit some specific network infrastructure, for example when legit MiM, *e.g.,* with a TLS company proxy, has been put in place between the server and the voter.

State-of-the-art threat models. We show in Table 1 the threat models we consider here and the threat models under which the Belenios protocol was proven secure; the FLEP claims to share a lot of similarities with Belenios [19]. (We recall that Belenios itself is derived from Helios [5].) A quick comparison between them in Table 1 shows that our attacks break FLEP under (even weaker) threat models for which a competitive protocol is (formally) proven secure. To compare with other protocols, Helios shares the same threat model as Belenios for verifiability and Civitas [10] and the Swiss Post protocol [34] are aware of the necessity to distrust a unique voting server and implement multiple servers/components to distribute the trust.

All these elements show that a channel attacker (and even more so a voting server attacker) is a reasonable threat model under which verifiability and ballot privacy are expected to hold.

B Attacks

B.1 Verifiability Attacks

Attack [Drop]: This attack variant of Attack [Replace] just drops Bob's ballot under the same assumptions, instead of dropping and replacing it. The attack proceeds as follows:

Step a: Alice casts her vote as expected. In this first step, the attacker executes honestly, *i.e.,* $H_1^{s_i} = H_1^c$ for all $i \in \{1,\dots,4\}$ and σ_1 is honestly computed. Hence, all the checks that Alice can do to be sure that her ballot b_1 has been included in the ballot-box succeed.

Step b: Bob follows the protocol but the channel attacker intercepts all messages after the creation of the ballot b_2 and computes by itself the responses that are normally sent by the voting server to the voting client as follows: $H_2^{s_1} = H_2^{s_3} = H_2^c$ but $H_2^{s_2} = H_2^{s_4} = H_1^c$ and $\sigma_2 = \sigma_1$. That is the attacker provides a receipt that refers to Alice's ballot. We do not assume that the attacker knows the signing private key sk_S to compute σ_2 since the attacker can simply replay the values obtained at Step a. The

[21] https://www.diplomatie.gouv.fr/fr/services-aux-francais/ voter-a-l-etranger/resultats-des-elections/article/elections-legislatives-resultats-du-premier-tour-pour-les-francais-a- l-etranger

[22] Note that the attacker would have to set up a malicious but legitimate-looking website thanks to having the valid certificate for it and combine this with DNS poisoning for instance. We stress that we are not assuming here that the attacker performs a phishing attack, the compromised certificate allows the attacker to "genuinely" impersonate the voting server to voters.

value H_2^c can be computed by the attacker since it is only made of public data (e.g. electionId and ballotBoxId) or data sent by Bob, such as b_2.

Step c: The attacker then blocks all the communications with the voting server, which will then time out and not register any ballot for Bob.

As opposed to Attack [Replace], there exists a non-suggested approach for Bob to detect this removal: Bob can refer to the signing sheet and see that there is no signature next to his name. Indeed, Bob did not send any ballot from the server point-of-view, the voting process has been stopped before its end. Bob could thus theoretically detect that something went wrong. We do think that it is unlikely that Bob does this complex verification (see Remark 11). Anyway, Attack [Replace] does not suffer from this and is completely stealthily.

At the end of this scenario, it is unlikely that somebody detects the removal of Bob's ballot and yet, the election result will not include Bob's ballot b_2 which has never been added to any of the ballot-boxes. Alice and Bob are convinced that their ballots have been counted (Bob is wrong) and the voting or the 3rd-party server will always receive consistent data. Therefore, the attack also shows that the FLEP fails to guarantee individual verifiability (despite the use of a 3rd-party).

Remark 10 (On the impossibility of detection using receipts)**.** *We stress that only the voting client knows the genuine and honest cryptographic values associated to the intended cast ballot b and the reference H^c. However, those values are freed and lost forever at the end of the session, when the voter downloads the PDF receipt, closes the browser tab, or logs out. Since the voting client fails to do the checks properly and since all the checks performed from the outside (from and with the 3rd-party, by observing data displayed to the voter, any internal checks the voting server could perform etc.) would succeed as they cannot know which ballot Alice or Bob intended to cast, no one can detect the manipulation based on the receipts. As we shall see, this is also the case for the other attacks we found. We come back to this discussion about detection at the end of Section 4.3.1.*

Remark 11 (Access to the signing sheet)**.** *As mentioned above, this attack is detectable if Bob has access to the signing sheet. This happens if he decides to come in person at the polling station to see the signing sheet or if he does an official query to the Consulate and then goes physically visit the consulate. The signing sheets were accessible this way only at the Consulate during a period of 10 ten days after the election.*[23] *The voters would thus need to come there to perform the check which is in total contradiction with the usability aim of e-voting. After this period, the signing sheet is no longer accessible to voters*[24].

In practice, assuming voters can detect the attack seems unrealistic: first, a large majority of the voters (>75%) decided to vote remotely. It is expected that they did so for convenience since consulates can be very far away. Why would then go to the consulate to consult the signing sheet? An attacker can thus safely guess that voters using the FLEP will not consult the signing sheet afterwards. The probability of this guess being correct can be highly increased based on naive social analyses to guess which voters are very unlikely to request the signing sheet.

B.2 Ballot Privacy Attacks

We give a full description of the [Swap$_{ID}$] attack variant that does not require the vulnerability (V1), in particular, how it guarantees that at least one ballot for each voting option is eventually stored in $\mathtt{u}_1$ and $\mathtt{u}_2$.

Attack [Swap$_{ID}$], swap ballotBoxId **by a voting server without relying on (V1).** To evade all possible detections, we recall the reasonable additional assumptions about other voters presented in Remark 6 (where $\mathcal{V} = \{v_1, \ldots, v_k\}$):

1. There are $E \geq k+1$ eligible voters in $\mathtt{u}_2$.
2. The attacker *knows* k voters, $V_1, \ldots, V_k$, that are different from Alice and such that V_i is willing to vote for v_i (in $\mathtt{u}_1$ or $\mathtt{u}_2$). Those voters are either colluding with the attacker or are honest but the attacker has a good guess about how they will vote.
3. There *exist* k other voters, $V_{k+1}, \ldots, V_{2k}$, different from Alice such that V_{k+i} is willing to vote for v_i in $\mathtt{u}_1$ for all $i \in \{1, \ldots k\}$. In contrary to $V_1, \ldots, V_k$, we do not assume that the attacker knows $V_{k+1}, \ldots, V_{2k}$, we solely assume that they exist. In practice, the attacker can just convince himself they exist based on statistical data (*e.g.,* previous election results). For example, almost all of the ballot-boxes from the 2022 election got at least one vote for each of the candidates
4. The attacker *knows* $E-1$ voters $V'_1, \ldots, V'_{E-1}$ eligible in any consulate (of the same election as Alice's) and how they are willing to vote. Those voters are either colluding with the attacker or are honest but the attacker has a good guess about how they will vote.[25]

[23] This is by lawful requirement: https://www.legifrance.gouv.fr/codes/article_lc/LEGIARTI000027572205

[24] This is by lawful requirement: https://www.legifrance.gouv.fr/loda/article_lc/LEGIARTI000023882783

[25] Strictly speaking, if those voters are not colluding and are eligible in a consulate $\mathtt{u}$ different from $\mathtt{u}_1$ and $\mathtt{u}_2$, then a similar assumption as 3. should be made about $\mathtt{u}$, that is it is expected that all voting options will be voted for, excluding those V'_i.

Those assumptions are reasonable in practice due to the existence, for each constituency, of both a ballot-box $\mathtt{u}_2$ with very few eligible voters (dozens) and a ballot-box $\mathtt{u}_1$ with a large number of voters (thousands).

Under those assumptions, a channel attacker can mount the following attack using the move technique [Move$_{ID}$]. We make explicit the two voters (Alice and Bob in the definition) whose ballots are swapped.

Step a. Alice's ballot (intended to be cast in $\mathtt{u}_1$) is swapped with an arbitrary voter *Bob* of $\mathtt{u}_2$ different from V_j for all j so that Alice's ballot b_1 is added to $\mathtt{u}_2$ and *Bob*'s in $\mathtt{u}_1$.

Step b. For all $i \in \{1,\dots,k\}$, if V_i votes in $\mathtt{u}_1$ then his ballot is moved to $\mathtt{u}_2$ with a swap with an arbitrary voter V' of $\mathtt{u}_2$ different from V_j for all $j \neq i$ and V'_j for all j. These swaps ensure that during the tally, $\mathtt{u}_2$ contains at least one vote for each voting option $v_1,\dots,v_k$.

Step c. Other honest voters (maximum $E-1$ voters) might want to cast a ballot in $\mathtt{u}_2$. Each of those ballots will be swapped with the ballot of one of the V'_i. Because the attacker knows the votes of $V_1,\dots,V_k$ and of $V'_1,\dots,V'_{E-1}$, the attacker will know the votes of all ballots in $\mathtt{u}_2$ except for Alice's ballot. Therefore, the tally of $\mathtt{u}_2$ will inevitably leak the Alice's plaintext vote since it is the unique unknown vote in the result for $\mathtt{u}_2$.

Thanks to the voters $V_1,\dots,V_k$ and Step b., the result for $\mathtt{u}_2$ is not inconsistent for the honest voters who wanted to cast a ballot in $\mathtt{u}_2$ since all voting options are expressed at least once. Thanks to the assumed existence of the voters $V_{k+1},\dots,V_{2k}$, the result for $\mathtt{u}_1$ is not inconsistent in the point of view of Alice, who wanted to vote in $\mathtt{u}_1$, since it contains a vote for each voting option too (including Alice's). Finally, any swap is balanced out so that there is no inconsistencies between the signing sheet and the ballot-boxes that can be detected.

Remark 12. *One may argue that this attack is invalid under the assumption of an honest voting client since* ballotBoxId *is received and processed by the voting client. We believe deciding what should be considered a malicious behaviors of the voting client and what is not is directly related to the question of what can be externally audited (we discuss this notion and how FLEP is made auditable in Section 5). If a piece of data that is received, sent, or computed can be externally audited as "valid" or "malicious", then an honest voting client would only accept valid ones. We argue* ballotBoxId *is not such a piece of data since it is a voter-specific piece of data, as opposed to the public key of the election, the JavaScript code, and the HTML templates, which are the same for the whole election. External auditors would have to know in which consulate voters are eligible and would have to test out all (or at least a significant ratio of all) voters. We thus consider that this attack does not require a malicious voting device and is thus in scope and valid.*

B.3 More on the Impact of our Ballot Privacy Attacks

Some extreme cases. In the 2022 French legislative election, some consulates received a unique ballot. *E.g.,* SVX-EKATERINBOURG received a unique (electronic) ballot during the first round among the 23 eligible voters.[26] This ballot expressed a vote for the candidate Catya Martin. In this extreme case, this directly leaks this unique voter's vote. Our attack additionally allows a channel attacker to choose the voter whose vote will be revealed.[27] For the sake of illustration, this unique ballot may actually not correspond to one of the 23 eligible voters but could have been moved from one of the consulate in the same constituency (*e.g.,* SYD-SYDNEY) which contains 98 853 eligible voters. Without claiming it happened, we stress that a channel or a voting server attacker had such a technical ability without leaving any evidence we could exploit now to know if the attack happened.

Why are our privacy attacks completely stealthily? Let us review all checks that are performed and could be done by motivated actors.

We already addressed the voting client checks in Remark 10.

A (honest) voting server is also unable do detect our privacy attacks. In particular, we recall that, all (tampered) data received and processed by the voting server are genuine- and honest-looking, the voting server does not know how voters intended to vote.[28]

We now reason about public information and independent verification services proposed to the voters. In particular, we assume that the integrity of the ballot is guaranteed by the 3$^{\text{rd}}$-party, independent researchers who operate a server executing an open

[26] All the other eligible voters abstain or use paper-based voting. It should be noted that the results are announced per ballot-box and per voting facility (e-voting, paper-based).

[27] We note that this additional leak is not desired: Alice's thought she was voting in a consulate with many voters to protect her privacy (the expected *anonymity set* was as large as the expected number of expressed votes).

[28] We also note that in the stronger threat model of a voting server attacker, checks performed at the voting server, such as the individual verification service offered by the FLEP voting server, provide no guarantee or security.

source program, named VVFE[29]. Our conclusions are as follows:

- publication of the result: unlike the two previous simplified variants, all the manipulated ballot-boxes contain (at least) one ballot for each voting option. They are the votes cast by voters $V_1, \ldots, V_k$ in $\mathtt{u}_2$ and $V_{k+1}, \ldots, V_{2k}$ in $\mathtt{u}_1$. Therefore, even if Bob's ballot has been moved from $\mathtt{u}_2$ to $\mathtt{u}_1$ (or conversely) then he will always see in the result at least one vote which corresponds to his intent, and it could be his vote. Bob cannot detect that his ballot has been moved away.

- universal verifiability: VVFE checks the wellformedness of the ballots. It consists in the check of the ZKP π. Because their context does not include the consulate (i.e. ballot-box) identifier $\mathtt{u}_1$ or $\mathtt{u}_2$ but only the constituency (i.e. election) identifier `elec`, the ZKP remains valid even if a ballot is moved from a consulate to another inside the same constituency. The attack cannot be detected based on these checks.

- individual verifiability: a voter can use his receipt to verify that his ballot has been counted. To answer these queries, VVFE recomputes all the references H and hashed values hb of the ballots provided by the authorities. Then, when a voter inputs their reference H or their seal cSU in the web service, VVFE looks for a reference or a hashed value that matches the entry. It is important to note that this look-up is performed taking all the ballots of the given constituency identifier `elec` into account. A more accurate look-up based on the consulate identifier $\mathtt{u}_1$ or $\mathtt{u}_2$ could be done when the voter inputs his seal but VVFE does not implement the check this way. This design choice is consistent with the properties that VVFE must ensure: moving a ballot from a consulate to another does not alter the (integrity of) the result of the election. VVFE has not been designed to detect privacy attacks. Finally, both Alice and Bob can query the 3rd-party with the receipt they own and VVFE will find a matching entry in its database. All those checks will succeed despite the manipulations performed by the attacker. The attack cannot be detected based on these checks either.

  Finally, the voters themselves are unable to detect that their ballot reference they were shown ($H_1^{s_2} = H_1^{s_4}$) are not computed with the right ballotBoxId (and the genuine, honest ballot b the voting client has had computed). Indeed, to be able to detect this, the voters would need to recompute H and thus to know what is hashed in H. However, voters do not know the ballot b in H since b is never displayed to the them.

C Reverse the Protocol

C.1 Reverse Methodology

We provide additional information about our reverse engineering workflow.

Big picture of the flow of exchanged messages. We show in Figure 4 all the POST and GET requests in a typical voter's journey. We also use the description of the voter's journey published on the government website to help voters with the voting process (see Figure 6 in Appendix C.2) to assign the different URLs and HTTP files from Figure 4 to the process steps from Figure 6. We then inspect the different requests and responses. One of them is depicted in Figure 5. Some of the fields of those requests can be related to the partial specification [19] (*e.g.,* `bulletin`) but many are omitted (*e.g.,* `hashBulletin`, `bulletinTemoin`) or not fully described (*e.g.,* `hSKeySU` which seems to be a public signing key after investigation).

That said, this is already enough to identify the main HTTP requests sent to the sever and associate them to the different steps of the protocol presented in the official documentation given in Figure 6:

1. *step 1 to 2:* a request to `pages/identification.htm` sends the login/password of the voter (and many other metadata) for authentication;

2. *step 2 to 3:* this transition does not require any request to the server. Step 3 displays the voter's choice to the voter and ask for confirmation to improve the user experience;

3. *step 3 to 4:* a request to `pages/envoiCodeActivationVote.htm` is done to the voting server to initiate sending the activation code by email;

4. *step 4 to 5:* surprisingly, two requests are made to the voting server. The fist one to the page `pages/verification_hash_bulletin.htm` and the second one to `pages/generic_vote.htm`. We will comment on the purpose of these two requests below, based on a better understanding of the fields of the different requests.

[29] https://gitlab.inria.fr/vvfe/vvfe

Status	Method	Domain	File	URL	Type
200	GET	votefae.diplomatie.gouv.fr	identification.htm	https://votefae.diplomatie.gouv.fr/pages/identification.htm	html
302	POST	votefae.diplomatie.gouv.fr	identification.htm	https://votefae.diplomatie.gouv.fr/pages/identification.htm	html
302	GET	votefae.diplomatie.gouv.fr	prevote.htm	https://votefae.diplomatie.gouv.fr/pages/prevote.htm	html
200	GET	votefae.diplomatie.gouv.fr	generic_vote.htm	https://votefae.diplomatie.gouv.fr/pages/generic_vote.htm	html
200	GET	votefae.diplomatie.gouv.fr	CheckConnexionElecteurServlet	https://votefae.diplomatie.gouv.fr/servlet/CheckConnexionElecteurServlet	plain
200	POST	votefae.diplomatie.gouv.fr	envoiCodeActivationVote	https://votefae.diplomatie.gouv.fr/pages/envoiCodeActivationVote	plain
200	POST	votefae.diplomatie.gouv.fr	verification_hash_bulletin	https://votefae.diplomatie.gouv.fr/pages/verification_hash_bulletin	plain
200	POST	votefae.diplomatie.gouv.fr	generic_vote.htm	https://votefae.diplomatie.gouv.fr/pages/generic_vote.htm	html
200	GET	votefae.diplomatie.gouv.fr	CheckConnexionElecteurServlet	https://votefae.diplomatie.gouv.fr/servlet/CheckConnexionElecteurServlet	plain
0	GET	votefae.diplomatie.gouv.fr	recepisse.pdf?election=3	https://votefae.diplomatie.gouv.fr/pages/recepisse.pdf?election=3	

Figure 4: All GET and POST requests of HTML and plain data of a voter's journey. *"envoiCodeActivationVote"* can be translated to "sendVoteActivationCode".

```
_target3             = nothing
_finish              = ok
_page                = 3
bulletin             = <ballot>
bulletinTemoin       = <ballot'>
cryptoLibrary        = tomwu
clientInfos          =
idElection           = 3
empreinteSuffrage    = [A4cxm+tMtlQF5s/SVn4OYgetjkgVB2sfNboY/wCd1PS=]
_csrf                = <CSRF>
hashBulletin         = [2xbqiw23fl6y7mfwx97l9oig782bcgdsdr6ws06t2xbqiw23fl6y7mfwx97l9oic]
```

Figure 5: Details of the POST 200 request at `https://votefae.diplomatie.gouv.fr/pages/generic_vote.htm`. Elements `<foo>` are replaced by placeholders such as `ballot`. Elements in `[bar]` are of the same format and length as the original ones but have been modified for protecting the voter's privacy.

C.2 Reversed Specification

We provided additional material that is useful to understand the overall workflow of the FLEP. We depict in Figure 7 the PDF receipt voters obtain at the end of the voting process. We also show in Figure 6 the user interface with the different steps of the voting process.

Remark 13 (On dynamic analysis.)**.** *The methodology we presented in Section 3.1 is purely based on semi-manual static analysis of the code and of the logs. We did try to conduct passive, dynamic analysis of the JavaScript programs by running them in a sandbox. However, we quickly found out that anti-sandbox mechanisms were in place and prevented us to run the code without immediate failure. We located the anti-sandbox code but we were unable to defeat it, which was not a problem in the end since we were able to extract the specification as explained above.*

D Other Concerns

We comment here different vulnerabilities or weaknesses in the FLEP. Most of them relate to attacks already known in the academic literature about e-voting. Our aim is not to provide a detailed description and discussion about each attack (original papers are cited for this purpose). Instead, we just want to recall that they exist and mention fixes that could be implemented. Note that those are not always exploitable in FLEP but yet deserve to be fixed. We discuss the challenges with implementing some of those fixes in the real-world setting of the FLEP. All these weaknesses have been discussed with the EFA Ministry, ANSSI, and Voxaly Docaposte.

D.1 Malleability of the ZKP

ZKP are complex cryptographic primitives that must be carefully used. In particular, as mentioned with the Fix 3, it is very important to think about which data should be included in the context of the proof to prevent attacks relying on tampering with

the ZKP context. We present two well-known attacks which do not directly apply to the FLEP but that show weaknesses in the FLEP that are easily fixable.

Micro-ballots re-ordering. In [18], Cortier and Smyth describe a verifiability attack against the e-voting protocol Helios (the ancestor of Belenios) which shares a lot of similarities with the FLEP. In order to understand how this attack works, we must detail how votes are encrypted and ZKP created. We deliberately omit many details about the FLEP which are not relevant to understand the attack.

As in Helios or Belenios, a ballot in the FLEP does not contain a unique encryption and a unique ZKP as presented in Section 2.1. Instead, the desired functionality is obtained by the means of several micro-ballots (one for each voting option) made of an encryption of 0/1 (chosen/not chosen) and one overall proof to ensure that the ballot is valid (*e.g.,* only one voting option has been chosen). Concretely, assuming a referendum with a simple "yes/no" question, a ballot is of form $b = (c_1, \mathsf{zkp}_1, c_2, \mathsf{zkp}_2, zkp_{all})$ where c_1 and c_2 are encryptions, zkp_1 and zkp_2 are ZKP that ensure that c_1 and c_2 encrypt 0 or 1, and zkp_{all} is a ZKP which ensures that at most one voting option has been chosen (i.e., c_1 and c_2 do not both encrypt 1). The attack exploits that the order of c_1 and c_2 is not included in the context of the ZKP. The ballot $b' = (c_2, \mathsf{zkp}_2, c_1, \mathsf{zkp}_1, \mathsf{zkp}_{all})$ is thus a valid ballot. This malleability weakness could be exploited by a channel attacker in combination with the vulnerability (V1) to modify the choice of a voter (if b codes for "yes" then b' will code for *no*) even though the attack [Replace] is more efficient and works under the same threat model. In [18], this weakness was exploited to replay a ballot that is different but that votes for the same choice. This is impossible to do in the FLEP due to the use of `tokenId`, see Appendix D.2.

The FLEP is vulnerable to this malleability weakness. To remedy this problem, the order of the micro-ballots must be reflected in the ZKP. For instance, we could include the voting option in the context of the individual ZKP, in zkp_1 and zkp_2.

Maliciously chosen parameters. In [16], Cortier *et. al.* show how it is possible to maliciously choose a group generator and a public election key to generate a ZKP that will always be valid, regardless of the encrypted value. This vulnerability would immediately falsify the verifiability property: given the structure of the ballots presented above, this vulnerability enables an attacker to forge a ballot that, when tallied, will actually count as an arbitrary number of votes for a target voting option. Coming back on the previous example, this would mean that an attacker is able to forge a ballot $b = (c_1, \mathsf{zkp}_1, c_2, \mathsf{zkp}_2, zkp_{all}))$ in which c_1 encrypts $n \geq 2$. As explained in [16], this attack can be fixed by adding the group generator and the public key of the election in the context of all the ZKP. These are two public elements.

Impact and fixed for FLEP. Because the FLEP implements cryptography based on elliptic curves and the curve is fixed in the voting client, it does not seem to be vulnerable to this attack. Remains the election public key that could be maliciously chosen. Further investigations would be necessary to decide whether this could be exploited or not. These investigations might be complex and error-prone, so we instead recommend implementing the aforementioned fix proposed in [16] that is as easy as implement as Fix 3, which has already been implemented by the vendor.

D.2 Ballot Replay Attack

Ballot replay attacks are one of the well-known attacks against ballot privacy. They have been shown applicable to Helios by Cortier and Smyth in 2011 [18] and their concrete impacts have been recently quantitatively studied by Mestel *et. al.* [28].

The attack is quite simple: the attacker replays Alice's ballot to amplify its impact on the result and thus learns more information than expected about Alice's plaintext vote. For instance, assume an election with 3 voters, Alice, Bob, who are honest, and Charlie who is dishonest. If Charlie can replay Alice's ballot then he can be sure that the winner of the election is the candidate Alice voted for.

Even if this attack seems to have a rather limited impact, Mestel *et. al.* [28] show that a very small number of replays (e.g., only 10) can significantly undermine ballot privacy in real-world elections.

A fix for FLEP. The `tokenId` is a random number generated by the voting server and sent to the voter to create the ballot[30]. We think that it could be used to perform *ballot weeding*: guarantee that all the `tokenId` which appear in accepted ballots are unique. In practice, if the voting server receives a ballot that includes a `tokenId` which is already used in another ballot, then it rejects it. The voter receives a new and fresh `tokenId` and re-votes.

[30] It would seem preferable to generate the `tokenId` in the voting client to prevent external manipulations but this different implementation choice does not seem to impact the security.

This check seems to be enough in the FLEP to prevent ballot replay. In the scenario presented above, the attacker would not be able to replay Alice's ballot. To do so, the attacker must modify the ZKP, which is not possible since the attacker does not know the random number used to encrypt the vote.

Remark 14. *This weeding process can be performed by the 3rd-party: if two ballots contain the same* `tokenId` *then the voting server must have misbehaved. This external verification would be an interesting safeguard. This check should be implemented in a future version of the VVFE (i.e., the open-source implementation of the 3rd-party service). Voxaly Docaposte confirmed that they have planned to implement this thanks to our findings.*

D.3 Private Key Generation

The manipulation of the decryption keys is a security concern of main interest to avoid arbitrary decryptions and thus preserve the confidentiality of the votes. Interestingly, the Code électoral (Article R176-3-1) precisely defines the quorum to conduct electoral operations (which include decryption).

> The electronic voting committee shall validly deliberate only when at least four of its members are present, including at least one of the holders or substitutes mentioned in item 5. [...]
> The substitutes may take part in the deliberations of the electronic voting committee even in the presence of the holders they are replacing. In this case, they have a consultative vote [only].
>
> —Code électoral, Article R176-3-1

This article defines 8 full members (holders) and their corresponding substitutes. They correspond to the 16 decryption authorities mentioned in Section 2.1 who each owns a unique share of the decryption key. The 8 holders (and their respective substitutes) are respectively from: ANSSI, 1 representative of the EFA Ministry and its CIO (chief information officer), Ministry of the Interior (similar to the Secretary of Homeland Security in the USA), Council of State, and 3 representatives of the French citizens abroad (including the president of the Assembly thereof).

The EFA Ministry interpreted this article to derive the following three operational criteria that define the quorum:

1. there must be 4 authorities to decrypt;
2. 1 of the 3 representatives (or their substitutes) of the French citizens abroad;
3. a holder and their substitute cannot both contribute to the decryption at the same time.

Only item (1) is cryptographically ensured, thanks to the threshold encryption scheme which is implemented with a threshold set to 4. Items (2) and (3) are not, but operational rules are supposed to ensure those rules are fulfilled. However, those rules can be bypassed since their satisfaction rely on the honesty of the quorum. In particular, a dishonest quorum made of only the EFA Ministry representative and its CIO holders and their substitutes reaches the threshold of 4 and is able to decrypt any single ballot at will.

We thus wonder whether all the criteria, including (2) and (3) could be cryptographically guaranteed. Can we share the decryption key among authorities to prevent any decryption if the quorum is not met. We generalize this into a new research question that we argue is of general interest in Appendix E.2.1.

D.4 (weak) Eligibility

The last weakness is about *eligibility*, a well-known difficult to ensure property. Its purpose is to guarantee that all the accepted ballots have been cast by legitimate voters. In the FLEP, this property closely relates to the authentication mechanisms used to identify voters. Three elements are used to this purpose:

- the login, a 12 characters long string received by email;
- the password, another 12 characters long string received by SMS;
- the activation code, a 6 digits code received by email.

These random codes are sent to the voters through 2 independent communication channels (email and SMS) as required by the Code électoral Article R176-3-7 and the security objective n°2-07 of the recommendations of the CNIL. The independence between the two channels allows to assume that they are not both compromised at the same time, and thus, at least one correctly authenticates the voter.

Two different subcontractors are in charge of respectively sending the logins (by email) and the passwords (by SMS), respectively Orange and mTarget. We have no information about how those logins and passwords are then sent to the voting server. We stress that in the case where a single subcontractor, such as Voxaly Docaposte, would collect both logins and passwords before sending it to the voting server, then this entity becomes an additional single-point-of-trust for the election result integrity since it could impersonate all the voters and vote on their behalf. This immediately falsifies eligibility under an honest voting server and channel. In the rest of the document, we are assuming this is not the case, that is the logins and passwords are directly sent to the voting server and that the two subcontractors Orange and mTarget do not collude.

Ballot stuffing under a voting server attacker. Assuming the channel attacker cannot eavesdrop the side-channels used to transmit the logins and passwords to the voting server, they cannot eavesdrop on those values and impersonate voters. However, a voting server attacker can always do so since it has the knowledge of all logins and passwords. This is due to a lack of trust distribution for the voter authentication process: a single entity (the voting server) is in charge of this authentication.

This attack can be detected in theory, even if hard to do in practice. Indeed, in order to maintain a strict correspondence between the signing sheet and the number of accepted ballots, the attacker needs to add a voter in the signing sheet for each ballot they cast, and thus have a guess and commit on who will not vote by using the FLEP or in-person at the polling station. In practice, the attacker strategy could be as follows:

- regularly during the election, the attacker commits on a voter who will not vote and signs the sheet on his name without adding a ballot for now. It is important to regularly commit because, unlike the ballot-box, the signing sheet is timestamped to monitor the participation rate. A significant increase just before the end of the election would look suspicious.

  Even though the voting server is in charge of writing in this signing sheet, some operational safeguards are in place to ensure its integrity. We are assuming here that the attacker is unable to tamper with the signing sheet, except for normal operations.

- if a commit appears to be wrong, *i.e.,* if a voter decides to electronically vote, then the attacker accepts the voter's ballot. The timestamp of the signature does not match with the date when the ballot has been cast but nobody can detect this inconsistency. Indeed, even if the signing sheet is timestamped in the database, it is not published online (we discuss later how voters can access the signing sheet and we shall see that it is fairly complex to do).

- if at the end of the election a commit appears to be true, *i.e.,* if the voter did not vote, then the attack can add an arbitrary ballot in the ballot-box right before the end of the election (the ballot-box is not timestamped and remains private until the end of the election, when it is finally transmitted to the 3rd-party). Ballot stuffing is thus possible.

Fixes are challenging to deploy. A simple theoretical solution to fix this weakness is to distribute the generation and the verification of the different codes (login, password, activation code). For instance, we could imagine that the EFA Ministry sends the login, the 3rd-party the passwords, and the service provider the activation codes. The 3rd-party then offers an API to the voting server allowing password verification.

Unfortunately, deploying such an infrastructure is difficult. The EFA Ministry and ANSSI are aware of this problem but did not found a satisfactory solution to overcome this single-point-of-trust problem. This weakness will probably not be addressed in the future version of the FLEP.

Based on this observation, it seems that academic research is still needed to develop solutions that match real-world constraints of deployments.

D.5 Honest Voting Device Assumption

Cast-as-intended is a well-known sub-property of verifiability. It ensures that a voter can be sure that her ballot contains her intended plaintext vote even if the voting device is compromised. The FLEP does not guarantee this property and thus considers verifiability properties with respect to an honest voting device.

Even if this assumption is standard in the literature (see for example [10, 15, 26]), we think it deserves to be discussed: in the FLEP, the voting device executes with the browser a JavaScript program sent by the voting server. Therefore, on the surface, it seems contradictory to assume an honest voting client but a compromised communication channel or even worse a compromised

voting server. Indeed, as far as we know, there is no built-in functionality in the browsers to check the integrity of a JavaScript file dynamically loaded.

This apparent contradiction is resolved in the literature with the notion of "auditability" of the code sent by the voting server to the voting client (see for example [7]). Indeed, this code is sent to anyone connecting to the election website and any cheating (a voting server sending a malicious JavaScript program) could be detected by comparing the received code with the honest, legitimate one. It is unlikely that voters are able to do such checks. For doing such checks, state-of-the art protocols assume a role of *auditors*, who are external auditors anyone (including tech-savvy voters) can execute. The idea is that if enough auditors are checking the consistency of the code sent by the voting server and if they are hidden enough among the voters, the voting server can no longer cheat since the risk of being caught becomes too high. As discussed in Remark 12, this assumption requires that the "voting client" is composed of static files only, i.e. files that are independent from the voter. This may make the design of the voting client more complex.

Auditability of the FLEP and recommendations. In the context of the FLEP, such external audits were made complex to conduct and never specified. We can provide a few insights to reduce the risk of attacks:

- unlike the current implementation choice, the system could decide to distribute the JavaScript program through a secure platform. This way, the integrity of the code would be protected by the platform. The platform could be a well-established website of a public authority or an application store. However, such a design choice has serious weaknesses: first, it requires that voters install a standalone software which severely impacts the usability of the system, and is quite expensive because development and maintenance costs[31]. Moreover, it puts an important trust assumption on the authority in charge of the distribution (public government, Google, Apple, etc.).

- if deploying the FLEP voting client as a standalone software is not an option, the voting client will be inherently vulnerable to integrity flaws. Fortunately, it is possible to dramatically increase the attack cost by improving the voting client auditability, as explained next. First, unlike the current FLEP implementation, the main voting client program logic and items to display must be included in static files that are easily comparable to honest, files of reference. Specifically for the HTML documents, they should be made of an uniform, static template in which holes correspond to a restricted set of user-dependent data. The system must then ensure all of the expected security properties regardless of how those holes are filled[32]. Ideally, the code would be distributed in the form of a Single-page Application (SPA)[33] as done by Belenios. This way, the voting client is made truly auditable.

  Moreover, unlike the current implementation, the data to be audited (JavaScript code and HTML templates) must be distributed prior to authentication (this is for free with the SPA framework). Indeed, to make the audits useful, the voting server should not be able to adapt the JavaScript code and the HTML templates it sends depending on the voter who is authenticating and who is trying to cast a ballot (or to audit the voting client).

  We recommend to at least implement these two proposals to improve the security of the FLEP. Similarly, we recommend the EFA Ministry and ANSSI to include them in the requirement lists for the next public call for tenders.

E Other Lessons

We presented our security analysis and its outcomes, as well as other concerns we found in the FLEP (discussed in Appendix D). Some important findings and discussions are relevant beyond the FLEP. In this Section, we step back and draw some lessons that are of general interest.

E.1 Voting Client as a Critical Component to be in Audit and Analyses Scope

The partial specification [19] has been used by the third-party to build their verifier service intended to let any voter independently check individual verifiability (*i.e.,* their ballot has been counted) and universal verifiability (*i.e.,* that the final count is correct). However, the voting client is not specified in [19]. Neither the design nor the implementation of the voting client are made amenable to analysis, even less accessible for public scrutiny. (We recall that the voting client code is obfuscated, see Section 3.1.)

[31] That said, there exist technologies such as Electron https://www.electronjs.org/ that dramatically lower the cost of deploying web applications as standalone software.

[32] Continuing Remark 12, it is now clear why checking the validity of ballotBoxId cannot be considered to be external auditors' tasks since it is a user-dependent piece of data. Hence the need for Fix 4 (or Fix 5).

[33] https://developer.mozilla.org/fr/docs/Glossary/SPA

Why the voting client is the most critical component for verifiability? Verifiability is a security mechanism that allows to dispense an e-voting system to having to trust the voting server, its administrators, the code it runs, etc. This is sometimes called *software independence*. As discussed in Section 2.2, the underlying relevant threat model when it comes to analyze verifiability is therefore a voting server attacker; more precisely, verifiability is supposed to make an e-voting protocol secure even when the voting server is *fully compromised*. This is the usual point of view in the e-voting literature and explains why the voting server attacker is the standard threat model for the election integrity (see Section 2.2).

From that point of view, the most important component of the protocol that should be the main target of audits and analyses is the voting client. If the voting client is securely designed and implemented, the protocol should ensure the election integrity under this threat model (see Table 1).

A very good illustration of this is that we have shown with our attacks that the partial specification and 3rd-party verification services are of no interest if the voting client is flawed as it was the case for the 2022 French legislative election (assuming a channel or voting server attacker). An implementation weakness in the voting client can completely defeat verifiability and privacy as we have shown. Therefore, the voting client must be fully open, documented, and audited.

That said, this does not dispense to make effort to secure the voting server, which seems to have been the main focus of the audits made on the FLEP. Indeed, securing the voting server from external threats reduce the attack surface and is beneficial to the overall system security. Our point remains: this is insufficient and misses the point of verifiability and software independence.

E.2 Reflect the Use Case Specificities in the Cryptography

E.2.1 Operational and Lawful Constraints

We have shown in Appendix D.3 that some lawful requirements governing when a quorum is met to *e.g.,* decrypt a ballot-box are not cryptographically enforced in the FLEP. In theory, fewer people than what is prescribed by the law can collude and decrypt a ballot-box, even though it does not constitute a valid quorum by law. To prevent such scenarios, some operational procedures (key storage in safes, use of secure envelops, etc) were put in place and are expected to be enforced by human safeguards (checks before openings).

Therefore, we ask the following question: is it possible to cryptographically enforce such operational constraints to prevent any human misbehavior? Are there solutions that are generic enough to be suitable for practically relevant constraints seen in real-world use cases?

Cryptographically enforcing quorum rules. More formally, we ask the following interesting research question:

Open question. *Given a set of authorities A and a valuation $\rho : \mathcal{P}(A) \to \{\top, \bot\}$, is it possible to share a decryption key among A's members such that for all $B \subseteq A$, B's members are able to decrypt if, and only if, $\rho(B) = \top$.*

This question is partly answered in Belenios specification [23] for specific quorum rules. The system allows to define mandatory authorities whose participation is necessary to decrypt. Regarding the quorum rules presented in Appendix D.3, this approach could be used, for instance, to cryptographically ensure item (2), *i.e.,* at least 1 of the 3 representatives of the French citizens abroad is present. Item (3), *i.e.,* holder and substitute cannot contribute at the same time, could also be cryptographically guaranteed based on a different approach: instead of creating 16 shares, the system could simply create 8 shares: one for each pair of holder/substitute member. Using the same share, a holder and their substitute cannot both contribute to the decryption at the same time.

These are two concrete solutions to reflect the quorum rules of the FLEP but are not generic enough to encode any arbitrary rule. The theoretical question remains open.

Remark 15. *Perfectly encoding the quorum rules in the cryptography would not seem appealing for the EFA Ministry and ANSSI in the short-term. They would like to conduct further risk evaluations and organize internal debates about the impact of the solution on the availability of the system. The EFA Ministry and ANSSI do not want to reach a situation in which results cannot be published on the d-day because the quorum is not met. Instead, they would like to keep the possibility of publishing the results and acknowledge that the quorum was not met in a public official document. They argue that the legality of this decision would then be contestable against the Court.*

Even if we think that this solution would be an improvement of the system, whether this should be implemented or not is a political decision.

E.2.2 Put the Whole Public Context in ZKP, Again

We have shown in Section 4.1 that a critical contextual information (ballotBoxId) was missing from the ballot ZKP. We also explained in Appendix D.1 that known weaknesses with missing contextual information in the ZKP possibly also impact the FLEP. This is has become a recurrent patterns of weaknesses with practical deployments of e-voting [16, 18, 24]. It is now recommended to include *by default* in the ZKP any public, contextual information about the election.

Our work shows that this pattern and the recommendation are still practically relevant in 2022.

E.3 Simpler is Better

We now argue why simplifying e-voting protocols are also beneficial to their security.

E.3.1 Defining and Simplifying the Voter's Journey

Beside coming up with a threat model specification, another challenge we had to solve was to determine what the voters should do and what they could do, especially for verification purposes. This distinction is important to assess if an attack will be detected, could be detected, or is actually undetectable. To resolve this, we argue it is crucial to precisely define the voter's journey and the system expectations about them. Ideally this description should be as precise as the protocol description. Before the election starts, the voters should receive a public document precisely defining what they are supposed to do and what are the guarantees they get if they do so.

Conversely, the system and its security analyses should not assume voters will do things they are not explicitly asked to do.

Cryptographic data and checks versus human voters. Taking the case of the FLEP as illustration, the voters receive different (confusing) information about the verification process:

- displayed on the last step voting web page:
  - (1) a first clickable link to "check the presence of the ballot in the ballot-box",
  - (2) a sentence informing the voter that, after the tally, the receipt can be used to check that their ballot has been taken into account.
- displayed in the PDF receipt:
  - (3) a clickable link to "control the reference of the ballot";
  - (4) a clickable link to the web service proposed by the 3$^{\text{rd}}$-party.

The voter may be confused by all these elements: what should I do? Which link should I use? What are the differences? Do I always get the same guarantees? Similarly, many different intimidating cryptographic data items are shown to the voters:

(a) the ballot reference H^{s_2} on the last step voting page, with no instruction to what to do with that,

(b) the ballot reference H^{s_4} on the PDF receipt "in order to control that your ballot is in the ballot-box",

(c) the seal cSU on the PDF receipt "in order to also check that your proof [the seal] of vote has correctly been produced by the system [...]",

(d) the ballot fingerprint hb^{s_4} on the PDF receipt "in order to check that the content of your ballot is the same throughout the election. This value should be checked against the one obtained when checking the presence of your ballot in the ballot-box".

Again, as a voter, what should I do with all that? What are the differences? Even if a FAQ is available[34], none of the these questions are precisely answered. We explain below how this voter's journey can be considerably simplified with only displaying the seal on the PDF receipt (c) asking to do the check (4).

[34] https://www.diplomatie.gouv.fr/fr/services-aux-francais/voter-a-l-etranger/elections-legislatives-2022/presentation-du-vote-par-internet/article/faq-du-vote-par-internet

Signing sheet. Another important aspect is the access for the voters to the signing sheet discussed in Remark 11. Indeed, even if this document is in theory accessible to each voter, the likelihood that a voter accesses this document is more questionable. Therefore, it appears as a key element for the security analysis to define whether the voter is requested to examine this document, is encouraged to access it, or is not supposed to check it. As we have seen, this impacts the security (more attacks are possible if the check of the signing sheet is not requested to voters). We stress that, to be on the safe side, we shall not assume voters will do what is not explicitly requested from them.

Simplifying the FLEP voter's journey. Related to the necessity of describing the voter's journey, it seems important to define it as simple as possible. The more complex the journey is, the less likely and the less often the voters will entirely follow it because of misunderstanding or discouragement.

Regarding verifiability for the case of the FLEP, it seems very important to use the 3rd-party web services to check the presence of the ballot in the ballot-box (check (4)). However, a voter may be confused and think that using the voting server services is enough (checks (1) and (3)). Moreover, even if the voter decides to use the 3rd-party services, she may be muddled by the fact that she can do a first check during the voting phase (where only the validity of the signature of the seal cSU is checked) but she actually needs to come back after the election closed to check that his ballot has been tallied by entering their ballot receipt H. These two verification steps at the 3rd-party, which add up to the two similar verification steps the voting server *also offers*, are highly confusing. They could be easily merged into a single one in which the 3rd-party logs all the seals it receives when voters verify during the voting phase. Once the voting phase ends and the 3rd-party receives the ballot-boxes from the voting server, the 3rd-party can verify that the ballots of all the seals received so far are indeed in the (right) ballot-boxes. As a result, we propose a modification of the FLEP that considerably simplifies it so that voters only have to store one cryptographic item (the seal cSU stored on the PDF receipt) and conduct one check, at any time they want. This simplification does not weaken the security of the protocol. On the contrary, it strengthens the security due to the simpler voter's journey (more voters would probably correctly perform the required check(s)).

Moreover, the simpler the voter's journey is, the easier the security analysis will be and less likely flaws occur. For instance, the first vulnerability is partly due to the redundancy between the different ballot references displayed to the voter. The vulnerability might have been easier to spot and prevent if only one reference was available in the PDF receipt generated by the server and on which no check is performed.

E.3.2 Simplify the Protocol

Related to the simplicity of the voter's journey, it seems also important to make the system as simple as possible to avoid flaws. In particular, it seems important to dissociate security and safety elements. The safety ones could be removed without altering the execution of the protocol.

Concretely, following this advice could have prevented the first vulnerability. Indeed, all the references but H^c are data that have been added for safety or robustness only. Indeed, they should only be used to check that the voting device and the server execute in the same context to detect early invalid ballots. In the FLEP, if the references H^{s_1}, H^{s_2}, H^{s_3}, and H^{s_4} are removed, then the protocol is no longer executable; the voter no longer receives a valid final web page nor receipt. It is likely that the vendor would have detected that the protocol implementation was flawed.

To some extent, the second vulnerability is also related to a safety element that is not correctly handled: the publication of the result per consulate. If these detailed results were needed for practicality and verifiability in paper-based voting systems, it seems useless for e-voting: the tally is now computable for a large number of ballots and verifiability is now achieved thanks to cryptographic data and no longer requires small ballot-boxes. Hence, the use of the per-consulate tally appears as an outdated feature that makes the system unnecessarily complex and introduces flaws. From an academic point-of-view, and based on the advice to make the system as simple as possible, we question the rationale of this legal basis for e-voting in France. However, it remains a political decision to allow or forbid this.

E.4 Transparency and Specification

We explain why e-voting systems should be made as open as possible with clear system and threat model specifications.

E.4.1 Need to Clarify the Threat and Trust Models

Based on our experience, it appears that there was no official document presenting the threat model and the security properties that the FLEP was expected to ensure. However, we think that it is a must-have for any e-voting system.

First, as shown in this work, such a threat model specification is required to conduct meaningful security analyses. In this work, we have defined a reasonable threat model that we argue is in line with the legal framework in France (Section 2.2). However, it would be more satisfactory that the FLEP designers or the responsible for the public call for tenders (EFA Ministry) provided such a specification themselves. Ideally, it could be made a lawful requirement as it is the case in Switzerland (see Remark 16), where clear security objectives and threat models are defined in Swiss law, as well as the requirement to provide cryptographic and symbolic proofs thereof.

As it is meaningless to assess the security of a system without a clear threat model definition, the lack of public specification of the FLEP is problematic. For instance, it is of little interest for the public to know that the FLEP has undergone audits without having access to the scope and the objectives of this audit. Since there seems to be no official threat specification, we wonder what was the considered threat model in this audit? Did the auditors have to do subjective interpretations as us? We believe that the absence of a threat model specification makes the audits (and the security analyses) more complex and error-prone. The auditors or the experts first need to interpret the informal and incomplete requirements before analyzing the system. As shown before, this interpretation is somewhat subjective and may depend on the people doing it (politics, experts, vendors, etc). This is a source of misunderstandings and confusions (*e.g.,* ambiguities about compromise of the communication channels and the server, or the definition of *ballot privacy*). These can be resolved through discussions with authorities, which, however, is time-consuming and does not scale. Overall, this lack of threat model specification is detrimental to the quality of the audits, security analyses by experts, and public scrutiny in general.

Second, the definition of the threat model and the security objectives are useful to the politics and the citizens to understand the security expectations of the system they shall use. We stress that publishing such a document can be done without disclosing industrial secrets about the system internals such as how the voting server is coded. Moreover, in the case of a call for tenders, the politics would be able this way to precisely compare the different solutions proposed by the different companies. This would make their choice more well-informed and objective.

Third, the vendors should be interested in a clear description of what their system must ensure to meet the expectations of the organizers of the call for tenders. This would give competitive advantages to the solutions that are the most well-thought in terms of security, which eventually incentivize the vendors to improve their products.

Remark 16. *As an illustration, consider the Swiss example. The Federal Chancellery has made the effort to formally define the threat model and the security properties that an e-voting system must ensure to be considered for political elections. This information is provided in the Federal Chancellery Ordinance on Electronic Voting (OEV)*[35].

Even if these requirements are too specific to immediately apply to the FLEP, this ordinance shows the feasibility of our advice for the French authorities. We hope that the next public tender, or better, a revision of the Code électoral will progress toward this direction.

E.4.2 Need for Transparency for Public Scrutiny

Last but not least, we generalize the above: transparency is a key criterion to choose an e-voting system for political elections. The FLEP has been reviewed by many entities (Voxaly Docaposte experts, ANSSI experts, external auditors, 3rd-party, etc.) and none of them found the vulnerabilities we disclosed in this document. Most likely because the scope of their intervention was too limited, the well-recognized experts in the academic e-voting community who run 3rd-party did not find them either, despite their privileged access to the system. Different factors can explain this.

First, an unsought lack of transparency between the vendor, ANSSI, and the auditors. We discussed the lack of a threat model specification. There might also be a lack of a system specification. While we stress that the publication of the partial specification [19] is an appreciated step towards openness, it is not enough to convince the system is secure. Its scope is too narrow (no description of the voting client) and it lacks key components (*e.g.,* threat model and security objectives specification). Most notably, as discussed in Appendix E.1, a complete specification of the voting client is lacking. As we have seen in Section 3.1, despite [19], the cost to obtain a complete description of the system is fairly high. All the information was somewhat accessible but hidden in a labyrinth of corridors and rooms.

From an academic point-of-view, security by obscurity is a no-go. We thus encourage the EFA Ministry and all the authorities who want to promote or deploy e-voting to push for more transparency. A good example is what happened in Switzerland after the discontinuation of e-voting in 2019: an e-voting system must be open access to match the legal requirements and the Federal Chancellery push for public scrutiny (see Remark 16). In the French context, the helpful discussions we had with the EFA Ministry and ANSSI are very encouraging and we believe in their good will to improve this state of affairs.

To conclude, we mention some concrete improvements that come with greater openness and transparency:

[35] https://www.fedlex.admin.ch/eli/cc/2022/336/fr

- prevent that the system is vulnerable to well-known attacks of the literature thanks to public scrutiny. A single team of experts cannot be aware of all the existing attacks but many different experts can. A full transparency of the system would encourage different experts to have a look to the system. Ideally, way before deployment, the organizers could launch programs like Public Intrusion Tests that promote public scrutiny with incentives like Bug Bounty and good communication for a general call for analysis to all experts (academia, hacking sphere, etc.). This has been done in 2019 and 2022 in Switzerland.
- increase the confidence that voters can have in the system. Full transparency allows to distribute the trust between many more experts, each owning a different expertise. For instance, we conducted a security analysis at the protocol level, but we cannot assert there is no other attack: we might have missed a vulnerability inside the scope of this analysis, and we have not looked for attacks outside the scope and from a different perspective (network-level vulnerabilities, implementation of the cryptographic primitives, etc.).
- discover new attacks and/or better understand the practical impact of recent ones. For instance, the privacy attacks presented in this document is similar to a recent type of attacks firstly presented in 2021 [11] against the Swiss Post e-voting system. Whether variants of those attacks also break other e-voting systems remains an open question.
- help the academic community to focus on the most practically relevant expected security goals and threat models and develop solutions which could be transferred from academia to the real-world (see for example Appendix E.2). This would promote communication between the two communities (practitioners and academia).

In summary, transparency is a win-win: vendors, politics, and voters with more secure systems and weaker trust assumptions and academic researchers to identify new practically relevant open research questions.

F Translations of the Main References

Almost all the quotes presented in this document have been translated from French documents and these translations would like to be as impartial as possible. However, to avoid subjectivity due to unconscious bias, we recall here the official French versions and our English versions.

Translations from the Code électoral [21].

Source	Original French version	Translated English version
Article R176-3-1, §9, §10	Le bureau du vote électronique ne délibère valablement que si quatre au moins de ses membres sont présents, dont au moins l'un des membres titulaires ou suppléants mentionnés au 5°. [...] [...] Les suppléants peuvent participer aux délibérations du bureau du vote électronique même en présence des membres titulaires qu'ils ont vocation à remplacer. Ils disposent alors d'une voix consultative.	The electronic voting committee shall validly deliberate only when at least four of its members are present, including at least one of the holders or substitutes mentioned in item 5. [...] [...] The substitutes may take part in the deliberations of the electronic voting committee even in the presence of the holders they are replacing. In this case, they have a consultative vote [only].
Article R176-3-9, §3	Le vote est protégé en confidentialité [et en integrité]	Votes must remain confidential
Article R176-3-9, §4	L'enregistrement du vote de l'électeur donne lieu à l'affichage d'un récépissé électronique sur le système de vote lui permettant de vérifier, en ligne, la prise en compte de son vote.	When a voter's vote is registered, the voter is provided with a digital receipt allowing them to verify online that their vote has been taken into account.

Translations from the CNIL recommendations [30].

Source	Original French version	Translated English version
Security objective n° 1-04	assurer la stricte confidentialité du bulletin dès sa création sur le poste du votant.	[the system must] ensure the strict confidentiality of the ballots as soon as created.
Security objective n° 1-07	assurer l'étanchéité totale entre l'identité de votant et l'expression de son vote pendant toute la durée du traitement.	[The system must] ensure that the identity of the voter and the expression of his choice can not be linked during the whole process"
Security objective n° 2-07	assurer la transparence de l'urne pour tous les électeurs. [...] Il s'agit de permettre aux électeurs de s'assurer que leur bulletin a été pris en compte dans l'urne et que les bulletins de vote sont construits de manière correcte.	ensure the transparency of the ballot-box for all the voters. [...] It must be possible for the voters to ensure that their ballot has been counted in the ballot-box.
Security objective n° 3-02	permettre la transparence de l'urne pour tous les électeurs à partir d'outils tiers.	The system must allow transparency of the ballot-box for all voters from third-party tools.
–	Niveau 3 : Les sources de menace, parmi les votants, les organisateurs du scrutin, les personnes extérieures, au sein du prestataire ou du personnel interne, peuvent présenter des ressources importantes ou de fortes motivations.	Security level 3: The threat actors include the voters, the election operators, outsiders, insiders within the provider or internal staff. They can be resourceful or highly motivated.

Translations from the Voxaly Docaposte specification [19].

Source	Original French version	Translated English version
–	Le numéro d'ordre de l'élection est ajouté à la configuration du bulletin (champ UUID), afin que ce numéro soit également pris en compte dans la preuve associée au bulletin. Cela permet ainsi de détecter éventuellement un bulletin qui aurait été déplacé d'une urne à une autre, ocasionnant alors un changement du numéro d'ordre de l'élection.	The election identifier [electionId] is included in the context of the ballot (field UUID), so that this identifier will be added in the [ZK] proofs associated to the ballot. This allows to detect if a ballot has been moved from a ballot-box to another, which would modify the election identifier.

G List of Changes

- v1.0, November 28, 2022: Initial version.
- v2.0, March 25, 2023:
  - Improved the privacy attacks, which are now possible under even weaker threat assumptions. In particular, the simultaneity of Alice's and Bob's votes in attacks [SwapC] and [SwapCs] is no longer needed. This makes the attacks more practical.
  - Improved presentation and clarity. In particular, with a focus on the main ideas in the core of the paper. All the details are still available in the appendices.
  - Update with up-to-date information regarding the next 2023 partial legislative elections using FLEP based on the new version of the specification [20] and private conversation with the stakeholders.

  The ballot privacy attacks have been slightly renamed to consider more explicit names. Here is the correspondence between v1.0 and v2.0:

  - SwapS $\rightarrow$ Swap$_H$
  - SwapC and SwapCs $\rightarrow$ Swap$_b$ (due to presentation choices, both attacks have been merged)
  - SwapID $\rightarrow$ Swap$_{ID}$

- v2.1, September 25, 2023: various improvements reflecting the camera-ready version of the paper published at Usenix Security 2023.

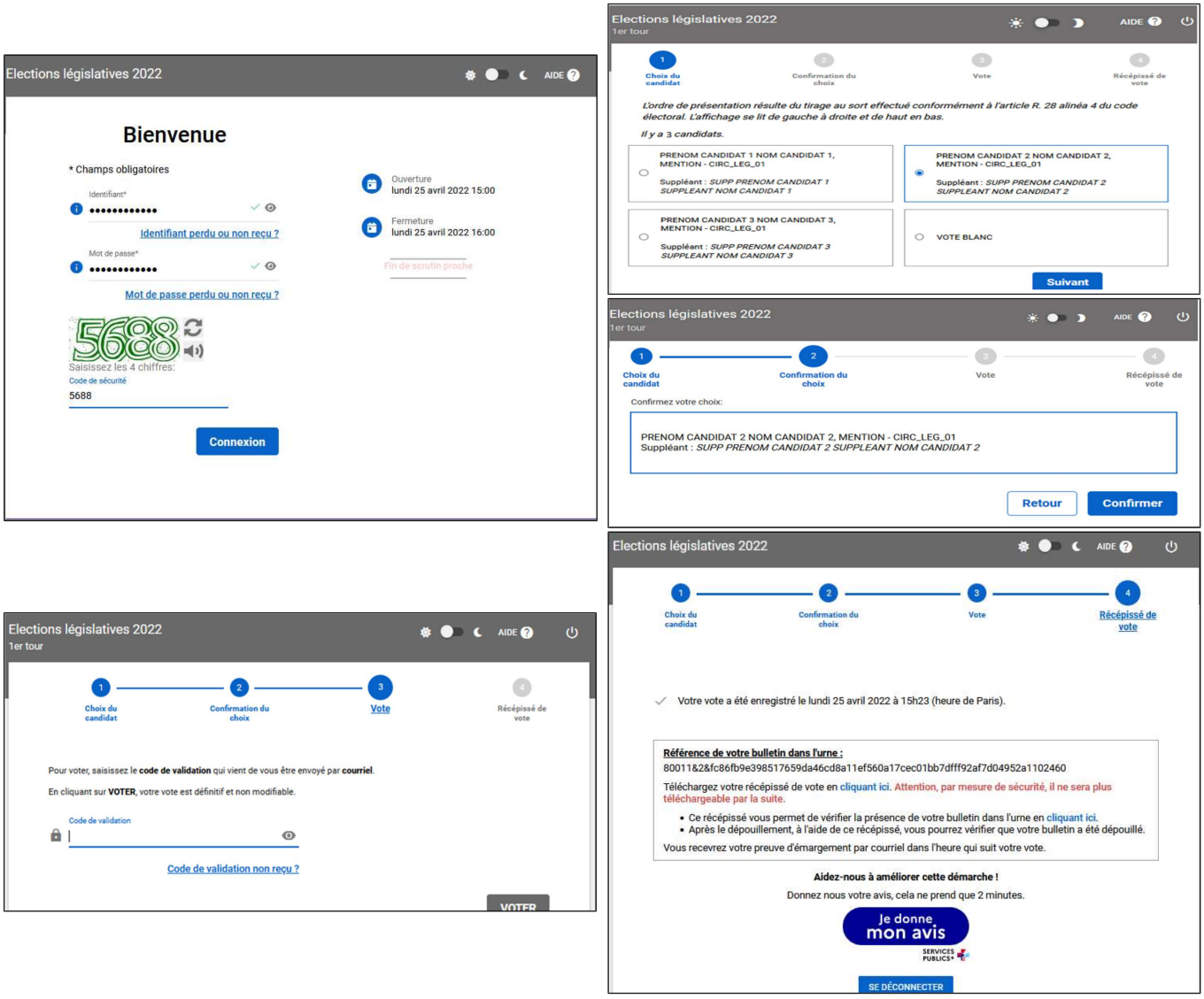
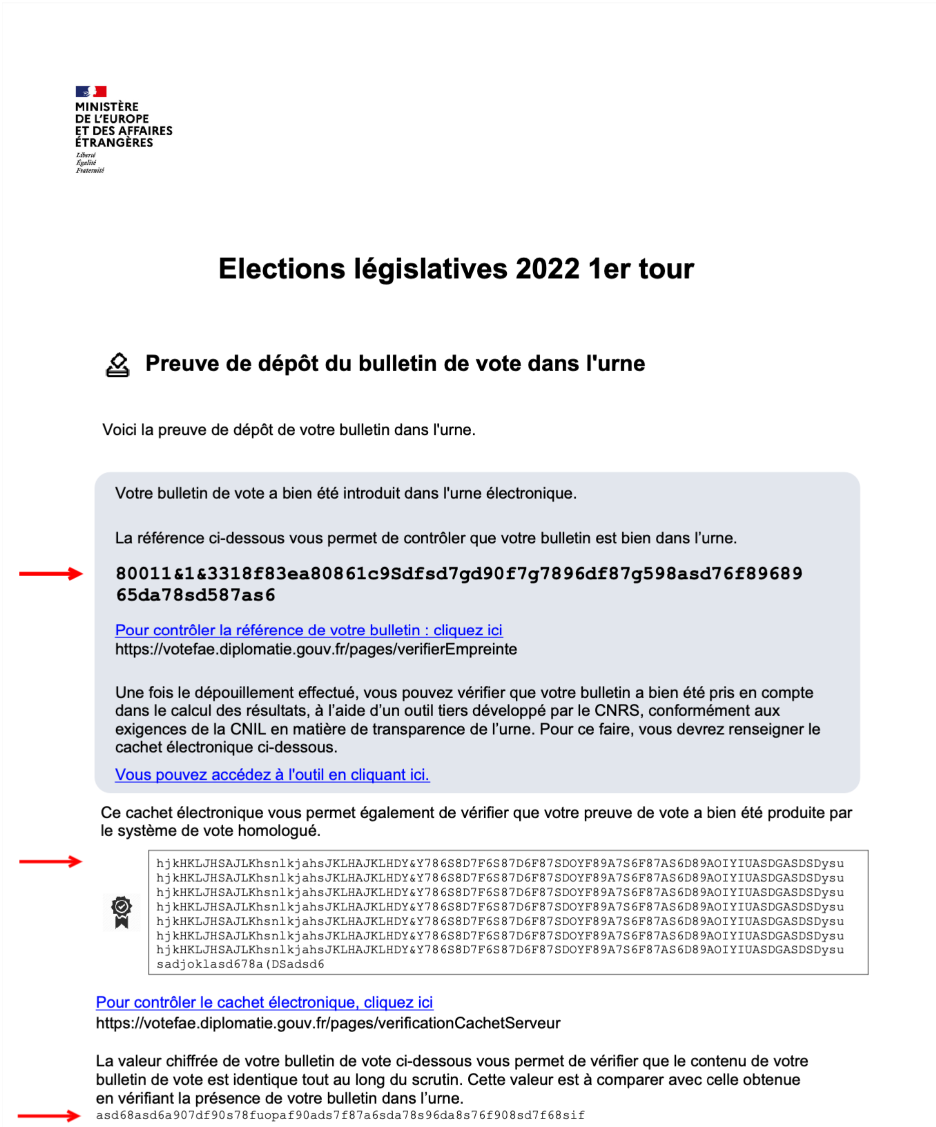


Figure 6: User interface of the FLEP with the different voter's journey steps (extracted from https://www.diplomatie.gouv.fr/IMG/pdf/presentation_du_portail_de_vote_cle4f6e8e.pdf). Step 1 (top left) is the login page. Step 2 (top right) is the vote selection page. Step 3 (middle right) is the vote confirmation page. Step 4 (bottom left) is the activation code prompt page. Step 5 (bottom right) is the final page that contains the ballot reference H^{S_2} and a one-use link to the PDF receipt (shown in Figure 7).



MINISTÈRE
DE L'EUROPE
ET DES AFFAIRES
ÉTRANGÈRES

Elections législatives 2022 1er tour

Preuve de dépôt du bulletin de vote dans l'urne

Voici la preuve de dépôt de votre bulletin dans l'urne.

Votre bulletin de vote a bien été introduit dans l'urne électronique.

La référence ci-dessous vous permet de contrôler que votre bulletin est bien dans l'urne.

**80011&1&3318f83ea80861c9Sdfsd7gd90f7g7896df87g598asd76f89689
65da78sd587as6**

Pour contrôler la référence de votre bulletin : cliquez ici
https://votefae.diplomatie.gouv.fr/pages/verifierEmpreinte

Une fois le dépouillement effectué, vous pouvez vérifier que votre bulletin a bien été pris en compte dans le calcul des résultats, à l'aide d'un outil tiers développé par le CNRS, conformément aux exigences de la CNIL en matière de transparence de l'urne. Pour ce faire, vous devrez renseigner le cachet électronique ci-dessous.

Vous pouvez accédez à l'outil en cliquant ici.

Ce cachet électronique vous permet également de vérifier que votre preuve de vote a bien été produite par le système de vote homologué.

hjkHKLJHSAJLKhsnlkjahsJKLHAJKLHDY&Y786S8D7F6S87D6F87SDOYF89A7S6F87AS6D89AOIYIUASDGASDSDysu
hjkHKLJHSAJLKhsnlkjahsJKLHAJKLHDY&Y786S8D7F6S87D6F87SDOYF89A7S6F87AS6D89AOIYIUASDGASDSDysu
hjkHKLJHSAJLKhsnlkjahsJKLHAJKLHDY&Y786S8D7F6S87D6F87SDOYF89A7S6F87AS6D89AOIYIUASDGASDSDysu
hjkHKLJHSAJLKhsnlkjahsJKLHAJKLHDY&Y786S8D7F6S87D6F87SDOYF89A7S6F87AS6D89AOIYIUASDGASDSDysu
hjkHKLJHSAJLKhsnlkjahsJKLHAJKLHDY&Y786S8D7F6S87D6F87SDOYF89A7S6F87AS6D89AOIYIUASDGASDSDysu
hjkHKLJHSAJLKhsnlkjahsJKLHAJKLHDY&Y786S8D7F6S87D6F87SDOYF89A7S6F87AS6D89AOIYIUASDGASDSDysu
hjkHKLJHSAJLKhsnlkjahsJKLHAJKLHDY&Y786S8D7F6S87D6F87SDOYF89A7S6F87AS6D89AOIYIUASDGASDSDysu
sadjoklasd678a(DSadsd6

Pour contrôler le cachet électronique, cliquez ici
https://votefae.diplomatie.gouv.fr/pages/verificationCachetServeur

La valeur chiffrée de votre bulletin de vote ci-dessous vous permet de vérifier que le contenu de votre bulletin de vote est identique tout au long du scrutin. Cette valeur est à comparer avec celle obtenue en vérifiant la présence de votre bulletin dans l'urne.

asd68asd6a907df90s78fuopaf90ads7f87a6sda78s96da8s76f908sd7f68sif

Figure 7: PDF receipt obtained at the end of the voter's journey. Red arrows were added by us. The first one points to the ballot reference (H^{s4}), the second to the signed seal (cSU), and the third to the ballot fingerprint (hb^{s_4}). The cryptographic values were modified by us to protect the voter's privacy.

.