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

Alexandre Debant, Lucca Hirschi
Inria

November 28, 2022

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 6 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. Our attacks are in the process of being prevented with our fixes for future elections. Beyond this specific protocol, we draw general conclusions and lessons from this instructive experience where an e-voting protocol meets the real-world constraints of a large-scale and political election.

Responsible Disclosure and Acknowledgments

We conducted this security analysis through passive analysis only; we never attacked voting servers. Therefore, we could not alter the integrity or the security of the election. Moreover, all the vulnerabilities reported in this document have been reported to the relevant stakeholders at least 3 months before publication. We thank those stakeholders, *i.e.,* Europe and Foreign Affairs French Ministry (EFA French Ministry), Agence nationale de la sécurité des systèmes d'information (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 French Legislative E-Voting Protocol (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.

Contents

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.,* [8, 13, 6, 1]). 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 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 1.1 million eligible voters. This was the largest election (in terms of expressed votes) worldwide using e-voting. In total, to elect 11 deputies, the French residents overseas have cast more than 524k ballots using FLEP.[1] They massively preferred the e-voting channel using FLEP (76%) over traditional paper-based voting (22.7%) or postal voting (0.3%)[2]. Due to the French legal framework, FLEP was used to organize eleven elections, one per constituency, each electing one member of the National Assembly. Precisely,

[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.

[2] 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

even if only the global result at the constituency level is used to elect a deputy, for each of the per-constituency election, ballots were aggregated by consulates in order to publish more fine grained result (per consulate).

As expected, the FLEP has high security ambitions. Some of those ambitions are related to lawful requirements (the "Code électoral" [11], that is the electoral code) and others correspond to the recommendations of the Commission nationale de l'informatique et des libertés (CNIL) [16][3] which are a list of requirements that such a system is expected to meet (even if there is no legal obligation). The FLEP has undergone audits by external auditors to assess that it meets those requirements, notably that it remains secure under strong threats such as internal threats. A partial specification has been published that describes the verifiability mechanism that is used in the protocol. To meet strong requirements, the election organizers have put in place a third-party, made of independent researchers, and suggested to all voters to visit this third-party web service and independently verify the presence of their ballots (and receipts).

Question. 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 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) in that: (i) an attacker in control of the server is also in control of the plaintext channel and (ii) a channel attacker has no control over the server's databases, bulletin board, authentication routines, etc.

Contributions. We contribute the following:

1. We reverse engineered the obfuscated JavaScript program running in the FLEP voting clients. Doing this on two publicly deployed or tested versions (test campaign at scale and the final, election-day version), we were able to cross-reference information in order to fill the several critical gaps in the public but partial specification of the FLEP. We obtained this way a full specification of the voting client and the verifiability checks. We also study the French lawful requirements and relevant recommendations and define a precise threat model specification that we argue is in line with the French legal framework and is also supported by the literature.

2. Analyzing this, we found two vulnerabilities that can be exploited by a channel attacker and even more so by a compromised server, 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 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.

3. 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) Selectively drop voters' ballots to defeat individual verifiability and thus modify the result of the election. We show how this is possible by creating multiple malicious receipts that are bound to a single ballot, all the voters receiving those will believe their ballot is counted while it has been dropped.

(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 the previous one 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 to another and observing the per-sub-election 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. In particular, voters receive receipts that seemingly correspond to the genuine sub-election. However, the attacker needs to decide which voters to target and attack when those cast their ballots. We also describe a variant of (A_3) that is possible under a voting server attacker and that is even easier to put in place.

[3] 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.

(We also discuss 3 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 could exploit now to know if they have been exploited. We responsibly disclosed those attacks to all impacted and involved actors: the operator EFA French Ministry, the vendor Voxaly Docaposte, and the institutional security advisor ANSSI[4]. We later disclosed our findings to the 3rd-party services supervisors. All of those 4 stakeholders have acknowledged and confirmed our attacks. Note that a comprehensive risk analysis of the new threats induced by our attacks is out of the scope of this paper.

4. We propose and discuss 5 countermeasures to those attacks. The EFA French Ministry, ANSSI, and the vendor (Voxaly Docaposte) have confirmed to us that they have used some of our countermeasures to fix the FLEP. They have confirmed that those fixes are expected to be implemented for future elections. However, their deployment will be subject to development timing constraints (implementation, testing, etc.) which may prevent the use of these fixes if new elections must be organized too soon. It is worth noting that such elections are likely to happen: the FLEP could be re-used for potential partial re-elections (because some elections were contested) and it could also be additionally re-used in case the French president decides to dissolve the assembly.[5]

5. We found other less-impactful but still concerning weaknesses in the FLEP. For instance, we noticed that the security of the FLEP could be improved by cryptographically enforcing the quorum rules for decryption. Currently, they are only guaranteed by a human consensus between members of the electronic voting committee. We also found that the FLEP is affected by known weaknesses such as ballot replay, incomplete Zero-Knowledge Proof (ZKP) contexts, etc.

6. Note that the FLEP is derived from the state-of-the-art academic protocol Belenios [8] that is affected by none of those weaknesses. Why is this so? 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. Those are of a broader interest.

   The partial specification published for FLEP has been used by the third-party to build their verifier service intended to let any voter independently check that their ballot has been counted and that the final count is correct. We show that such a partial specification and 3rd-party verification services are of no interest if the voting client is not fully open, documented, and audited. An implementation bug in the voting client can completely defeat (individual) verifiability and privacy as we show. The voting client should actually be a core target of audits and analyses.

   Operational constraints that are often omitted or not well-understood in our academic community are both a challenge and a opportunity to tightly reflect them in the protocol and the cryptography. For example, a practically relevant research question is how to cryptographically enforce the detailed rules defining the quorum for deliberations and tally. While many of those rules are often unique to a use case, most of them are of general interest and are worth studying from a mathematical point of view. Another example is the requirement in the FLEP use case for multi ballot-boxes. If Belenios offered such a feature, the FLEP would not have to add this feature itself and making the mistakes of forgetting a contextual information in the ZKP.

   We also advocate for simpler protocols and more importantly for simpler voter's journeys and verification tasks. Taking the FLEP as an illustration, we show how one could simplify the protocol to display 1 cryptographic data item instead of 4 and require voters to do one check instead of 4, while improving the protocol security at the same time.

   Finally, we promote transparency and openness and argue that more of those could have avoided the FLEP vulnerabilities and attacks.

Outline In Section 2, we present how we studied the French legal framework and how we reverse-engineered the FLEP to build a system and threat model specification. In Section 3, we present the vulnerabilities and attacks we found and discuss our fixes. We also discuss other less-impactful concerns. In Section 4, we draw more general lessons from this work, that go beyond the FLEP. We conclude in Section 5.

[4]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.

[5]This is rarely done, 5 times since the 60s. Note that the timeline for these elections and whether the fixes will be ready on time is not known for now. Nevertheless, the timeline for the publication of this work has been approved by the stakeholders.

2 Reverse the French Legislative E-Voting Protocol

The first step to conduct our security analysis was to gather enough information to precisely describe the FLEP and its security objectives. We distinguish three main sources of information:

1. the lawful requirements and the recommendations expressed by the CNIL [16] to define the different levels of security an e-voting system must satisfy depending on the importance of the elections and the risk involved. Obviously, the French legislative elections are subject to the highest level, *i.e.,* level 3, as claimed on the FLEP website[6].

2. the partial specification [19] published by the vendor (Voxaly Docaposte) for the explicit purpose of enabling external auditors and voters verify their ballot and the election outcome. In particular, it enabled the 3rd-party (*i.e.,* a team of French researchers, see Section 2.1.2) develop an external verifier, which is an explicit CNIL requirement.

3. the (obfuscated) JavaScript source code running in voters' web browsers collected by us and different voters during elections as well as in-situ observations about the election process from the voters' point of view collected purely passively.

By reversing the JavaScript voting client code, studying and cross-referencing the 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). As we shall see, this system specification is sufficiently precise and comprehensive to unarguably claim that it suffers from our vulnerabilities. All the impacted and involved actors (Voxaly Docaposte, ANSSI, EFA French Ministry, 3rd-party) have acknowledged our system specification is correct. Except for an aspect we could not infer from the outside (shown in blue in Figure 5) but that is irrelevant for our attacks, we obtained our system and threat model specification before our discussions with the stakeholders started.

2.1 Architecture

On April 21st 2022, the vendor of the FLEP product, Voxaly Docaposte, published a partial specification of the system [19]. Importantly, it describes how the FLEP is deployed in practice to address all of the specificities imposed by the organization of the French legislative elections.

2.1.1 Geographical Organization

French citizens overseas are gathered in 11 electoral regions (*e.g.,* north America, north Africa, Asia-Oceania, 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 part of a country. In the French legislative elections, each constituency elects one deputy. For each of the two rounds of the election, the n decryption authorities generate a unique public encryption key pk_E for all consulates and their corresponding private keys $\mathsf{sk}_{\mathsf{E}1}, \ldots, \mathsf{sk}_{\mathsf{E}n}$. Each of the 11 constituencies then run a sub-election identified by a unique election identifier $\mathtt{elec}_i$ cryptographically bound to each ballot.

A specificity of these sub-elections is that the results are not only published by constituency (which is anyway required to announce the elected deputy). The French law requires that the results are also published at the level of consulates. Therefore, it has been decided to associate different ballot-boxes, one for each consulate, to each sub-election. Each ballot-box is identified by an identifier $\mathtt{u}_j$. The final election result is obtained by aggregating all results from all ballot-boxes in a given constituency. Figure 1 sums-up the organization of these elections for a few constituencies and consulates.

2.1.2 Protocol Roles

The FLEP is similar to well-known protocols such as Helios [1] or Belenios [8] (which itself is derived from Helios). It thus distinguishes different roles we describe next. In the following, we will always refer to these 5 roles when detailing the threat model or the protocol itself. The 5 roles are:

- **voter:** they are the agent 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. The registration of these elements have been done before the election starts using services of the EFA French Ministry.

[6] https://www.voxaly.com/vote-par-internet-pour-les-francais-de-letranger-dans-le-cadre-des-elections-legislatives-2022/

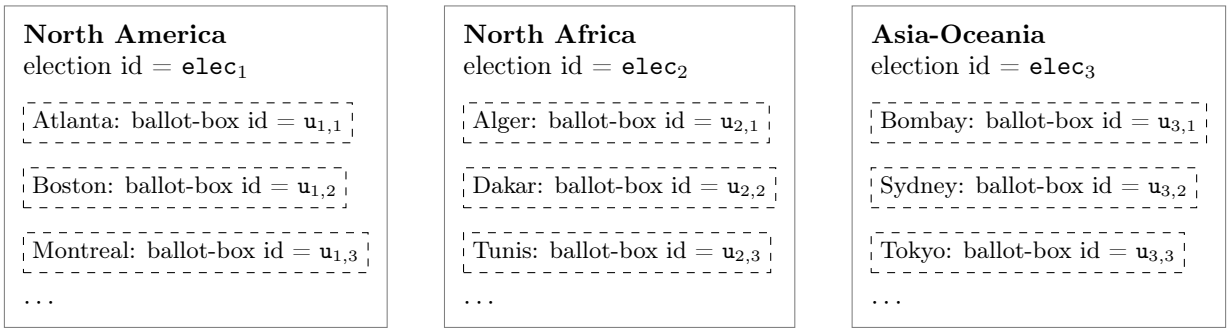


Figure 1: Organization of the French legislative elections. 3 of the 11 constituencies are depicted. Each of them runs its own election using FLEP to elect one deputy. The eligible voters of each constituency are grouped into consulates (*e.g.,* all voters in Atlanta) with their own ballot-box. Per-consulate results are also published but only the per-constituency result elects a deputy.

- **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 French Ministry whose purpose is to collect all the ballots. This server is also in charge of authenticating voters. It runs closed-source but audited Voxaly Docaposte software.
- **decryption authorities:** they are the authorities who can decrypt the ballots. More precisely in the FLEP, they are 16 authorities sharing a decryption key based on 4-threshold encryption scheme, that is only a group of at least 4 authorities can collude to decrypt.[7]
- **3rd-party**: it is an external server operated by independent researchers and engineers from a French lab (LORIA) and research institutes (CNRS, INRIA), on request of the EFA French Ministry. It provides to the voters some verification web-services[8] whose purpose is to ensure the integrity of the election: each voter can verify that their ballot appears in the ballot-box (individual verifiability) and the 3rd-party verifies that the tally has been correctly computed (universal verifiability)[9]. More details about these services will be given in Section 2.4. The source code of those services is open-source[10]. (Note that it is a CNIL requirement to propose such a third-party verification service (see CNIL Security objective n° 3-02 in Section 2.2).)

2.2 Security Goals and Threat Model

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 model that should be considered. Fortunately, there exist requirements that the FLEP should comply with, notably the Code électoral [11] (*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 [16]. As mentioned above, the FLEP is supposed to achieve the highest level of security defined by the CNIL, i.e., level 3.

Remark 1. *Even if fullfilling 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 compromise. This matches all the characteristics of a political election.*

[7] We will come back in Section 3.4.3 on this threshold and how the legal requirements for decryption could be better reflected in the cryptography.

[8] https://verifiabilite-legislatives2022.fr/

[9] https://verifiabilite-legislatives2022.fr/informations.html

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

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 A.)

2.2.1 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 [11]

> "[the system must] ensure the strict confidentiality of the ballots as soon as created."
>
> —CNIL, Security objective n° 1-04 [16]

> "[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 [16]

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 2. *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. *[4] or Bernhard* et. al. *[5]. 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 *in this document.*

2.2.2 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 will focus on the last property, *individual verifiability*, that we will show the FLEP fails to guarantee. Individual verifiability immediately relates to the Code électoral and the CNIL recommendations as follows:[11]

> 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 [11]

> 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 [16]

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

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 [16].

> The system must allow transparency of the ballot-box for all voters from third-party tools.
>
> —CNIL, Security objective n° 3-02 [16]

2.2.3 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 [16]

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.
- the voting server is operated by the EFA French Ministry which is not necessarily representative of the population. Having to entirely trust the EFA French 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[12] (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%[13] of voters.
- 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 [17, 2]. Again, having to trust this vendor would represent a massive trust shift for such political elections.

[12] 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

[13] 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

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).[14] (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.

Table 1 summarizes the assumed trustworthiness of all roles for the different properties. Note that it is acknowledged by public authorities that the FLEP does not provide *cast-as-intended* (voters have no guarantee that their voting device encrypts the intended vote). Therefore, Table 1 assumes a trustworthy voting device[15].

State-of-the-art threat models. We also show in Table 2 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 [1].) A quick comparison between Table 2 and 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 [**?**] and the Swiss Post protocol [18] 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 voting server attacker (and even more so a channel attacker) is a reasonable threat model under which verifiability and vote confidentiality are expected to hold.

	Voter	Voting device	Com. channels	Voting server	Dec. auth.	3rd-party
Verifiability	☺	☺	😈	☺*	☺*	☺
Confidentiality	☺	☺	😈	☺*	☺	☺

☺= trustworthy
😈 = compromised
☺*= trustworthy (However, compromise decreases attacks complexity.)

Table 1: Threat model under which our attacks break the FLEP security goals (details in Section 3)

	Voter	Voting device	Com. channels	Voting server	Dec. auth.	3rd-party
Verifiability	☺	☺	😈	😈	😈	☺
Confidentiality	☺	☺	😈	😈	☺	☺

Table 2: Standard threat model under which a state-of-the-art e-voting protocols such as Belenios **was** proven secure [8]. (There is no 3rd-party role in this protocol, however it assumes an honest public bulletin board. The latter can be achieved by a trusted 3rd-party; we thus assume the latter honest for comparison.)

[14] 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.

[15] We will comment on this assumption in Section 3.4.5.

2.3 Reverse Methodology

In addition to the architecture of the system and the threat model, the element needed to conduct our security analysis is a comprehensive description of the protocol. Unfortunately, [19] provides a partial specification only. Indeed, the claimed purpose of this document was to comply with the CNIL requirement security objective n° 3-02, that is to allow external auditors such as the 3rd-party role to develop verification services. We believe [19] **minimally** achieves this (providing the least information possible about the system) in order to not disclose confidential information or industrial secrets Voxaly Docaposte might want to keep. Hence, 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.

We describe here how we overcame this limitation in order to build a comprehensive description of the FLEP (see Section 2.4 and Figure 5) for which, as mentioned earlier, we have later obtained formal acknowledgment of correctness by the relevant stakeholders.

2.3.1 Obtaining Data

During the two rounds of the election, we collected all the web browser's interactions with the voting server during the different steps of the protocol thanks to a few eligible voters. These are gathered into HTTP Archive format (HAR) files that can be easily generated by browsers such as Google Chrome or Firefox. They log all the HTML, CSS, JavaScript, etc. files received by the browser together with all the HTTP requests and responses. For instance, we collected for one typical voter's journey 15 JavaScript files, 4 HTML files, 4 plain data exchanges, etc.

Remark 3 (On user data)**.** *Note that the only user data we collected are data received and sent from and to the voting server. Since vote confidentiality is supposed to hold under a compromised server, it must be impossible to extract how voters voted from those logs.*[16]

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

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

2.3.2 Reverse Engineering and Data Cross-referencing

Big picture of the flow of exchanged messages. We show in Figure 2 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 Section 2.4) to assign the different URLs and HTTP files from Figure 2 to the process steps from Figure 6. We then inspect the different requests and responses. One of them is depicted in Section 2.3.1. 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 (in Section 2.4):

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

[16] That is also true when taking our attacks into consideration since they require an active attacker while we only passively collected data.

```
_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 3: 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.

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.

Reverse engineering JavaScript voting client code. As mentioned above, some fields of the requests were not described in detail or not described at all in [19]. 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 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 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[17].

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[18]. 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 and the obfuscation was done differently. 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 with the production code base. An example of such a side-by-side comparison is depicted in Figure 4.

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 have the HAR). 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

[17] https://www.npmjs.com/package/js-beautify
[18] https://amsterdam.consulfrance.org/Elections-legislatives-2022-vote-par-internet-second-test-grandeur-nature

```
navclientApp.controller("PageVoteController", ["$scope", "$http", "$location", "$timeout", "
    breadCrumbService",
function(e, t, i, n, a) {      // core logic computing HashClient starts next line
  e.vote = function() {        // line 234 in scripts.js (after js-beautify)
        if (data.param.signatureEnabled && !e.aVote) {
            e.aVote = !0, e.erreurHashVerification = !1;
            var i = forge.md.sha256.create();
            i.update(e.bulletinCrypte + data.election.ordre + data.param.electeurEtOrdre);
            var n = i.digest().toHex(),
                a = function(e) {
                    [...]
                }(n),
                o = data.election.ordre + "&" + n + a;
            sessionStorage.setItem("HashClient", o);
```

Listing 1: Test Phase, `scripts.js`

```
function(e, t, n) {
  function ot(e) {
    [...]
    function v() {           // line 8980 of app.bundle.js (after js-beautify)
      return (v = Pe()(Re.a.mark((function t() {
        var n, r, a, l, u, c, s;
        return Re.a.wrap((function(t) {
          for (;;) switch (t.prev = t.next) {
            case 0:
              [...]
            case 3:          // core logic computing HashClient starts here
              return (n = new jsSHA("SHA-256", "TEXT")).update(o.bulletinCrypte + f.idTour + d
                  .ordre + f.electeurEtOrdre),
              r = n.getHash("HEX"),
              (a = new jsSHA("SHA-256", "TEXT")).update(o.bulletinCrypte + o.voteSignature),
              l = a.getHash("HEX"),
              u = f.idTour + "&" + d.ordre + "&" + r + y(r),
              sessionStorage.setItem("HashClient", u),
```

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

Figure 4: 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). (As we shall see, this will be critical information since `HashClient` will act as a receipt tracking the sent ballot; that is the ballot reference H^c described in Section 2.4.) Comments were added by us.
As we can see, the control flow of the test phase (Listing 1) is much simpler and allows to understand when this piece of code is triggered and what happens next (not shown here). This is much harder to see in the production phase code (Listing 2). However, only the production phase code matters in the end, notably because the protocol has changed and notably the content of `HashClient`. Therefore, we had to cross-reference both code bases.

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 a trustworthy 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 legitimate data honest voters would produce.

On dynamic analysis. So far, the presented methodology is purely based on semi-manual static analysis of the code and of the logs. We actually tried 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.

2.4 Reversed Specification

A full description of FLEP. We can now present a full description of the FLEP. In the following, we 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 5 and correspond to the voter's journey depicted in Figure 6.

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 $\mathsf{client\_info}$ (some meta-data about the voting client) and b_{temoin} 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 the HTML document `generic_vote.htm` displaying the different candidates $\mathcal{V}$ the voter can choose. This document also contains some hidden identifiers such as the election identifier $\mathsf{electionId}$ 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 confirms their choice by clicking on "Confirm".

Step 4: The voting client sends a request $\mathsf{envoiCodeActivationVote}$ ($\mathsf{sendVoteActivationCode}$ in English) to the voting server. Upon reception of this request, the voting server generates 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 computes some cryptographic material:

- 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 series of ZKP (π) proving that v is a legitimate choice of candidate, *i.e.,* $v \in \mathcal{V}$. Those proofs are also bound to `tokenId`, the per-session unique identifier received a step 1.
- The *hash value* $h := \mathsf{hash}(b \| \mathsf{roundId} \| \mathsf{electionId} \| \mathsf{ballotBoxId})$ where $\|$ is a delimiter, $\mathsf{roundId}$ is the round of the election (1 or 2), $\mathsf{electionId}$ is a per-election (hence per-constituency) unique identifier, and $\mathsf{ballotBoxId}$ is a per-ballot-box (hence per-consulate) unique identifier.
- The *ballot reference* $H := \mathsf{roundId} \| \mathsf{electionId} \| h \| \mathsf{errorCodes}$ where $\mathsf{errorCodes}$ are error correction codes for the whole message.

  This reference is computed and exchanged at different steps of the protocol, we specifically note H^c to refer to the ballot reference that is computed by the voting client and stored in the $\mathsf{SessionStorage}$ of the voting client browser[19].
- The *activation hash value*[20] $h_{\mathsf{code}} := \mathsf{hash}(b, \mathsf{code}_{\mathsf{activ}})$. 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 $\mathsf{code}_{\mathsf{activ}}$, bound to the ballot.[21] That said, the precise role of h_{code} is unimportant for the rest of the presentation.

The values b, h_{code}, and $\mathsf{code}_{\mathsf{activ}}$ are then sent to the voting server with the request $\mathsf{verif\_hash\_bulletin}$. The voting server then verifies the validity of the activation code $\mathsf{code}_{\mathsf{activ}}$, recomputes the ballot reference H, denoted by H^{s_1}; indeed H^c is not sent to the voting server. The response to the 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*[22] : $hb := \mathsf{hash}(b)$. The ballot b is again sent to the voting server, along with hb and h_{code}.

[19] The $\mathsf{SessionStorage}$ 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 redirection. See, for example, the documentation for Firefox: https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage

[20] Called *hashBulletin* in the voting client code.

[21] We note that this is completely inefficient since the activation code $\mathsf{code}_{\mathsf{activ}}$ 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 $\mathsf{code}_{\mathsf{activ}}$ can be quite easily brute-forced given h_{code} and b.

[22] Called *empreinte du choix* in the specification [19].

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 are 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 voters in the final step (we specify the seal cSU below).

Step 5: Finally, the voting client receives from the voting server an HTML document *generic_vote.html* 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 $<$ `pclass` $=$ "`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[23] that tests that H^{s_3} equals the ballot reference stored in the voting client $\mathsf{SessionStorage}$, that is the ballot reference H^c computed by the voting client at step 4. 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.[24] The downloaded PDF receipt (see an example in Figure 7) contains the ballot reference H, the seal cSU, and the ballot fingerprint hb.

Despite those values being (partially) specified in [19] to be specifically computed values and despite some of those values having counter-part in the voting client, which has itself computed H and hb, the values displayed in the PDF receipt are never checked by the voting client and could be arbitrary. 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

$$\mathsf{infoSU} = \mathsf{roundId} \| \mathsf{electionId} \| \mathsf{electionName} \| \mathsf{ballotBoxId} \| hb^{s_5} \| \mathsf{errorCodes},$$

$\mathsf{electionName}$ 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:

$$\sigma = \mathsf{sign}_{\mathsf{sk_S}}(\mathsf{hash}(\mathsf{infoSU})).$$

[23] Surprisingly enough, this test (and an associated error handling) is the only embedded JavaScript code (it is inlined in the HTML document, instead of being included as usually done). Also, this is the only JavaScript code that is not obfuscated at all. This could be evidence that this piece of code went through a different production process (*e.g.,* it could have been added later).

[24] In practice, for most modern browsers (Chrome, Firefox, Safari), the PDF receipt is downloaded and opened in the current tab, hence replacing the previous page. We will come back to the implications thereof in Section 3.2.2.

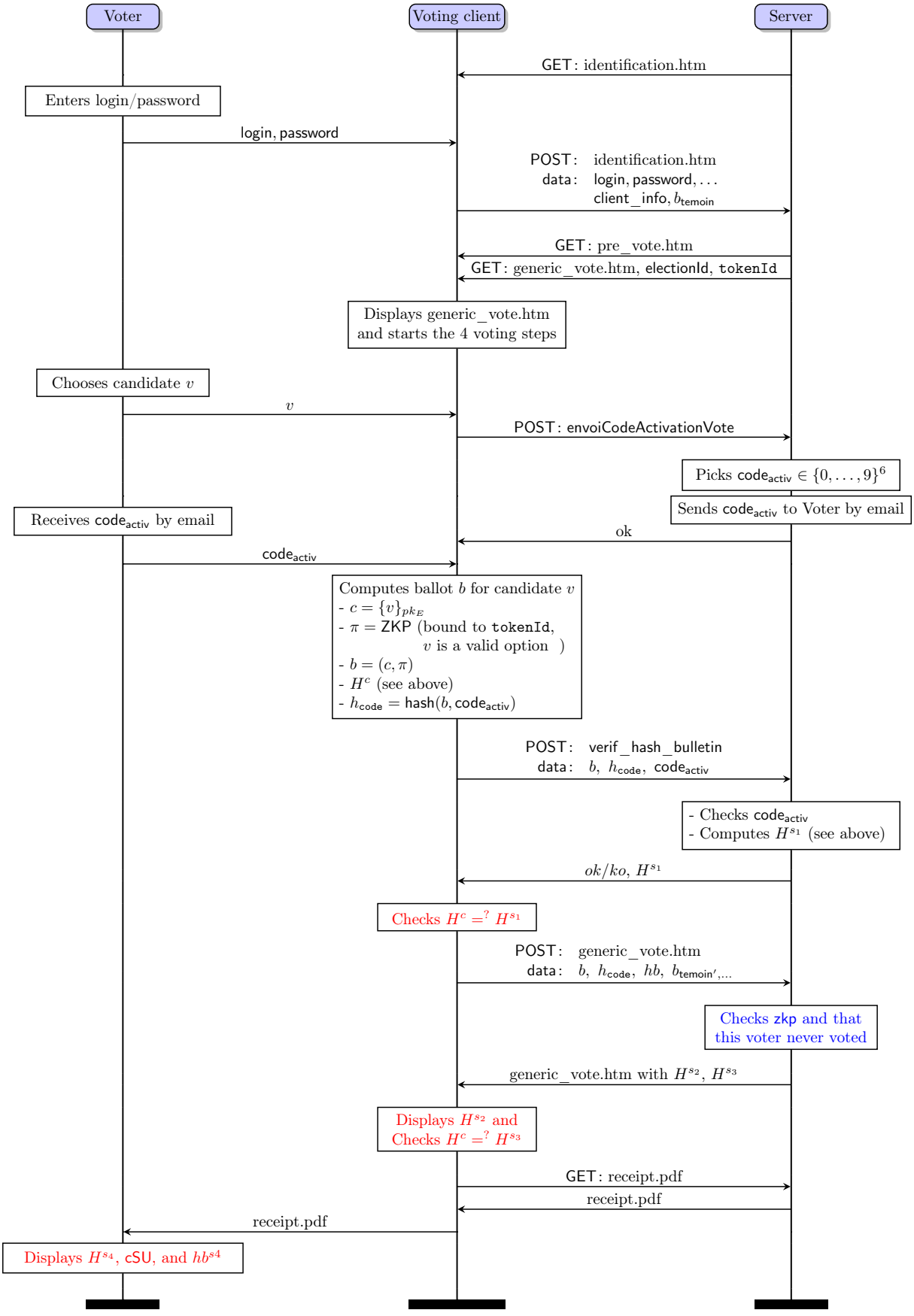


Figure 5: Description of the French Legislative Election Protocol. Actions in red will be of relevant importance in the vulnerabilities presented in Section 3. Actions in blue are elements obtained during discussions with EFA French Ministry, ANSSI, and Voxaly Docaposte.

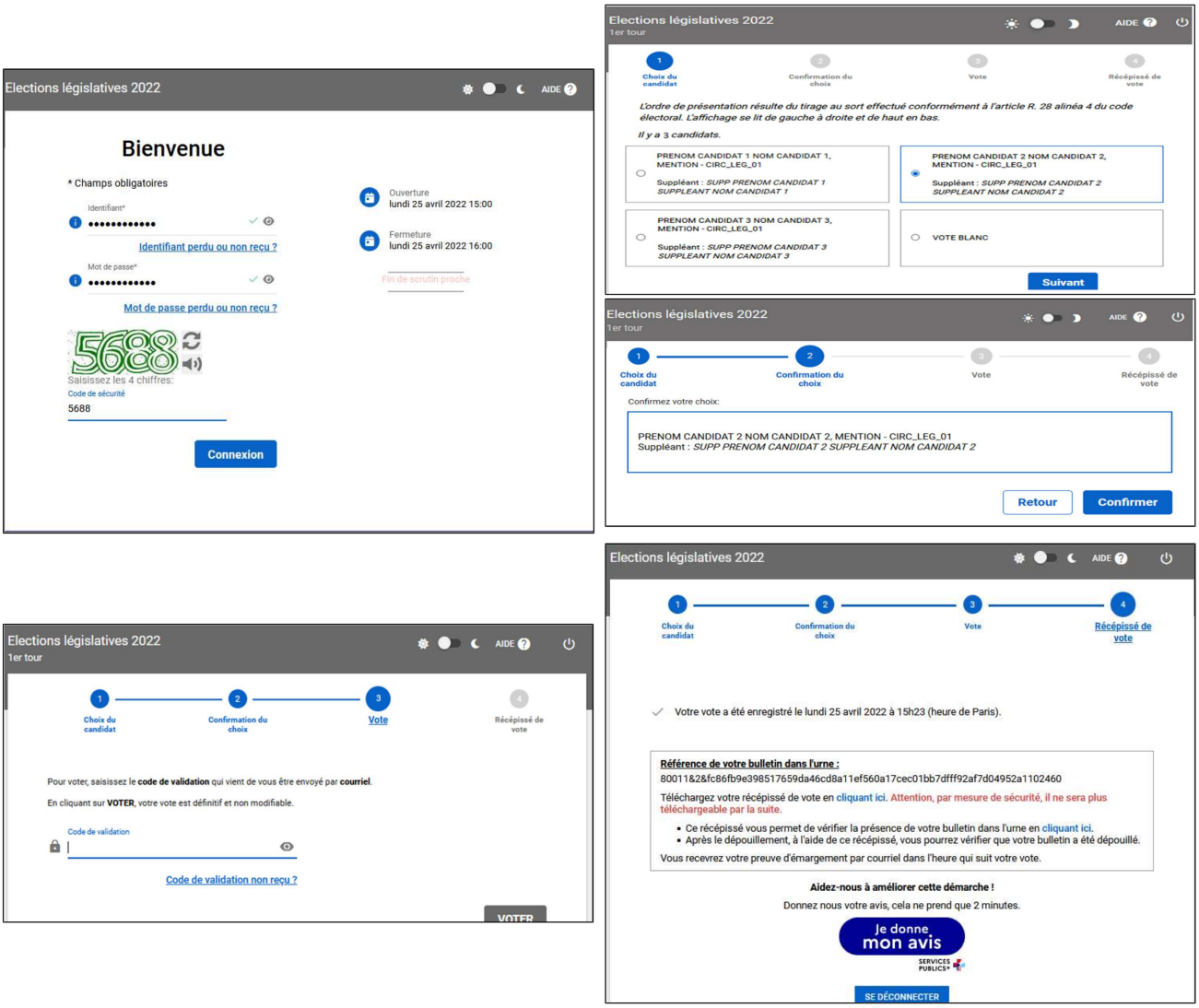
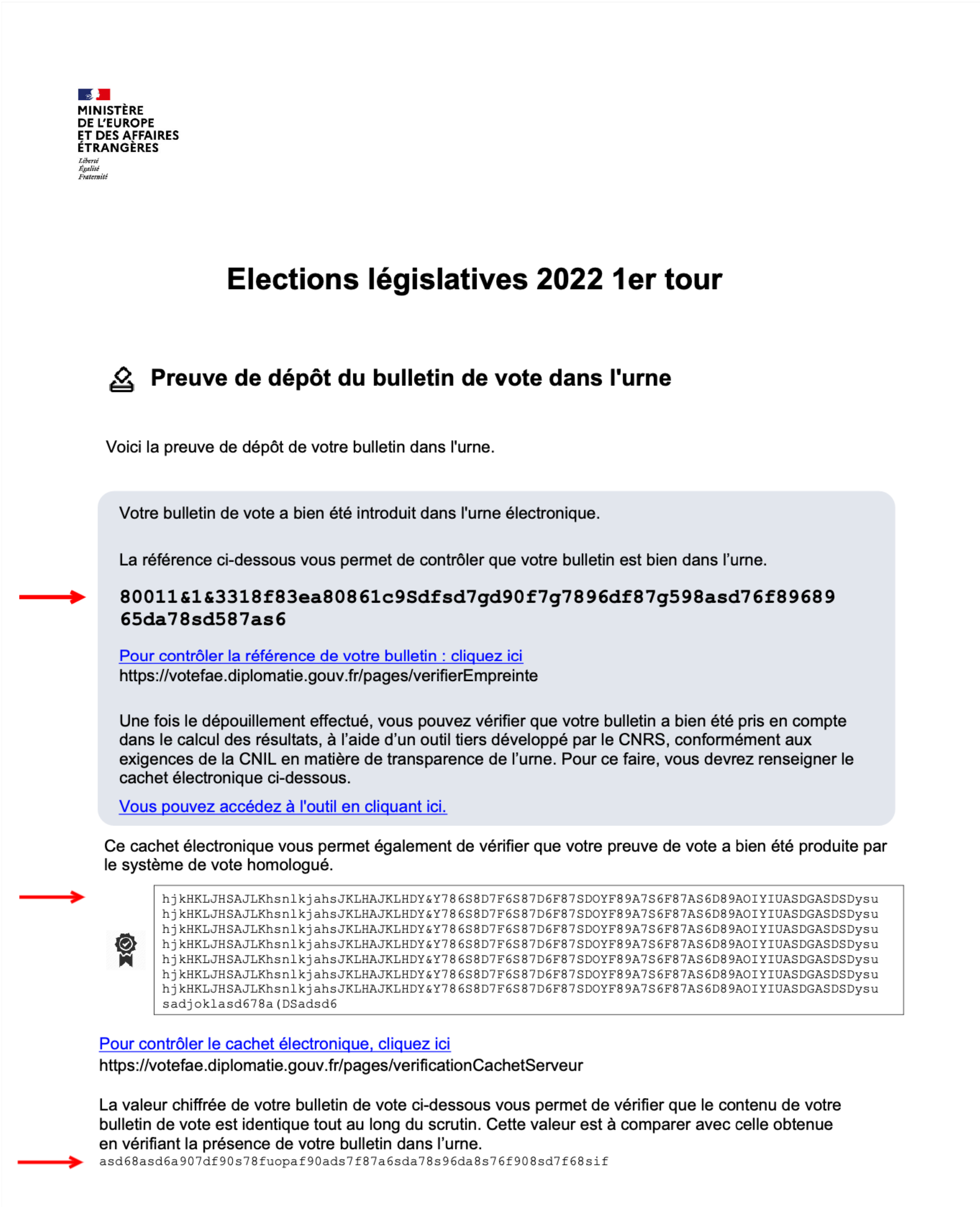


Figure 6: User interface of the FLEP with the different voter's journey steps (extracted from https://www.diplomatie.gouv.fr/IMG/pdf/presentation_du_portail_de_vote_cle4f6e8e.pdf). Step 1 (top left) is the login page. Step 2 (top right) is the vote selection page. Step 3 (middle right) is the vote confirmation 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
Liberté Égalité Fraternité

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.

.

3 Vulnerabilities, Attacks, and Fixes

We first present the two vulnerabilities we found (Section 3.1). We then show how these vulnerabilities can be exploited to break verifiability and the integrity of the election (Section 3.2.1) and the confidentiality of target voters' votes (Section 3.3). We also propose fixes to those attacks, some of them 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 French Ministry, and Voxaly Docaposte to be implemented in the mid-term.

Name	Attack on	Threat model	Impact	Fix that will be deployed
Drop	Indi. Verif.	Channel att.	Drop any cast ballot	Fix 1: display H^c
Replace	Indi. Verif.	Channel att.	Replace any cast ballot	
SwapS	Ballot privacy	Voting server att.	Learn some target voter's vote	Fix 4: add ballotBoxId to the ZKP
SwapC	Ballot privacy	Channel att.	Learn some target voter's vote	
SwapCs	Ballot privacy	Channel att., some voters collude	Learn any target voters's vote	
SwapID	Ballot privacy	Voting server att., some voters collude	Learn any target voters's vote	Fix 4 + Fix 3: 3rd-party displays ballotBoxId

Table 3: Summary of the attacks found. The last column presents one of our fixes that has been chosen by the stakeholders to be implemented in the FLEP in the mid-term.

3.1 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.,* 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 2 was necessary to identity those vulnerabilities, [19] only was not enough.

3.1.1 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 ZKP 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. In the JavaScript code, they refer to values that are expected to be equal and to correspond to the legitimate value H^c computed by the voting client thanks to various consistency checks (shown in red in Figure 5). Unfortunately, those are flawed in that they fail to enforce that the displayed references (H^{s_2} and H^{s_4}) are the same as the only legitimate reference computed by the voting client H^c.

This may happen as soon as the communication channels (or even worse the voting sever) are compromised. Indeed, as presented in Section 2.4, the voting client computes once the reference H^c that corresponds to his ballot, and receives 4 other references from the voting server, H^{s_1}, H^{s_2}, H^{s_3}, and H^{s_4}. According to Figure 5, the voting client performs consistency checks to ensure that:

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

Despite these checks, there is no guarantee about the values of H^{s_2} and H^{s_4}, which are the only references visible by the voters, respectively in the last web page (see Figure 6) and in the PDF receipt (see Figure 7). An attacker who controls the communication channels can take advantage of this implementation flaw to falsify the verifiability of the FLEP and thus modify the result of the election. We stress that the voters themselves

are unable to recompute the genuine, honest value of H (that is H^c) since they never learn the value of their ballot b, *i.e.,* it is never displayed to them.

3.1.2 V2: Malleability of Sub-Election Identifiers

As presented in Section 2.1, the FLEP is deployed in a complex environment. In particular, unlike usual academic e-voting protocols, an election is not associated to a unique ballot-box but several ones of heterogeneous sizes which are tallied individually. In the partial specification of the system [19], Voxaly Docaposte seems to be aware of this complexity and provides arguments to justify that this complexity is taken care of:

> "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".
>
> —Voxaly Docaposte, Partial specification of the system [19, p.11]

Unfortunately, we found out that this statement is incorrect. Indeed, as described in Section 2.1, 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. 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. An attacker who controls the communication channels can re-organize how ballots are gathered across the different ballot-boxes to learn information about a target voter's choice as we shall see in Section 3.3.

3.2 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.

3.2.1 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$_1$[Drop](breaks Indiv.Verif): This scenario requires at least two voters, Alice and Bob and assumes a channel attacker. In the following, we re-use all the notations introduced in Section 2.4 and Figure 5. 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 1:** Alice casts her vote as expected. In this first step, the attacker does not interfere in the protocol and the voting server executes honestly, *i.e.,* $H_1^{s_i} = H_1^c$ for all $i \in \{1, \dots, 4\}$ and cSU is computed as defined in Section 2.4. Hence, all the checks that Alice can do to be sure that her ballot b_1 (contained in H_1^c) has been included in the ballot-box succeed.

- **step 2:** Bob follows the protocol but the attacker interferes in the communications. Actually, the attacker intercepts all Bob's requests to the voting server 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 $\sigma_2 = \sigma_1$. 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 1. to Bob. 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 to the server such as b_2.

  Even if 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. Moreover, Bob can use the web services (provided by the voting server or the 3$^{\text{rd}}$-party) to be convinced that "his" ballot has been added in the ballot-box. Indeed, Bob got Alice's receipt which corresponds to a ballot added in the ballot-box at step 1. All the checks Bob was suggested or instructed to do succeed despite his ballot b_2 has been dropped.

There exists however a non-suggested solution to make sure that Bob detects 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 5). Anyway, we shall present a variant of this attack below that remedies this problem and is completely stealthily.

Therefore, at the end of this scenario, it is unlikely that somebody detects the removal of Bob's ballot[25] 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 FLEP fails to guarantee individual verifiability (despite the use of a 3rd-party).

Remark 4 (On 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 download 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 will also be the case for the other attacks we found. We come back to this discussion about detection at the end of Section 3.3.1.*

Remark 5 (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.[26] 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[27].*

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.

Attack$_2$[Replace](breaks Indiv.Verif): For the previous attack Attack$_1$[Drop](breaks Indiv.Verif), the attacker could only drop ballots. Because a ballot is not cryptographically bound to a voter, this previous attack can be highly improved as follows: during step 2, instead of just intercepting all the messages sent by Bob and not send anything to the voting server, the attacker can forge a new ballot $b_{\mathtt{att}}$ 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 $\mathsf{hash}(b_2)$. Hence, the channel attacker can not only drop ballots, but also replace them by attacker-chosen ballots. Moreover, this addition of a new ballot makes the attack undetectable: the signing sheet now contains a signature for Bob.

Remark 6. *Interestingly, Attack$_2$[Replace](breaks Indiv.Verif) is not possible if FLEP used the ballot format of Belenios, since those include voters' signatures. Therefore, the removal of signatures from Belenios to FLEP – as a mean to comply with CNIL recommendations (ballots should be anonymous)– without compensating this loss with another security mechanism introduced a weakness in the protocol.*

High level overview of the threat:

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

[25] In other rare circumstances, for example when b_2 was the **only** ballot in Bob's ballot-box to vote for a given candidate v, Bob can detect that b_2 has been removed by observing the election result (since there is not even one vote for v). This is extremely unlikely for large consulates (hence with many expected expressed votes in the ballot-box). Moreover, the attacker can use some other compromised voters to vote once for each of the possible candidates to completely avoid detection, as explained in Section 3.3.

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

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

3.2.2 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 the voter's journey but is more complex to implement.

Fix 1: display H^c instead of H^{s_2} on the last web page of the user interface (step 5). The reference H^c is computed by the voting client. Hence, we can be sure that it corresponds to the cast ballot and the candidate the voter intended to vote for (assuming a trusted voting client, see Section 2.2). If H^c was displayed on the last web page of the user interface (instead of H^{s_2}), 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 Attack_1[Drop](breaks Indiv.Verif) and Attack_2[Replace](breaks Indiv.Verif). However, it complicates further the voter's tasks, which are already quite complex (see Section 4.3.1). Moreover, new instructions to conduct those extra checks need to be explained to voters.

Remark 7. *We note that the manual consistency check between the reference H^{s_2} displayed on the last web page and the reference H^{s_4} printed in the receipt is made difficult and error-prone: some browsers (e.g., Chrome v.106.0, Firefox v.105.0.3) open the PDF receipt in the current tab and the FLEP system prevents the voter from re-loading the last web page (the one displaying H^{s_2}). A conscientious voter must carefully copy H^{s_2} before opening the PDF receipt otherwise the check becomes impossible. The voter's tasks may become too complex to be understood and accepted in practice.*

According to the discussion we had with the EFA French Ministry, ANSSI, and Voxaly Docaposte, the fact that the last voting page cannot be reloaded is an intentional feature. They claim it is necessary to satisfy the Security objective n°1-07:

> *"[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 [16]*

We understand the compliance constraint but believe that the data associated to the last voting page could be locally stored in the voting client for the lifetime of the session at least (for example in the SessionStorage*). This way, the voters could easily reload the page. The voters would be requested to click on a final "Log out" button to clean up the session storage. This would also transparently happen when voters simply close the tab where they voted.*

Fix 2: make the voting client generate the PDF receipt. Another solution is to let the voting client generate by itself the PDF receipt. Such a local computation reduces the risks related to a compromised communication channel, or a malicious voting server.

For the voting client 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 pk_S) and its content before generating the PDF receipt. Moreover, the voting client would have to load some additional JavaScript library to generate a PDF. However, loading libraries in a critical system introduce new threats. Are the libraries secure and free of trapdoors? This aspect must be carefully studied. An alternative solution would be to manually generate the PDF without relying on external libraries but using instead a predefined template made of predefined bitstrings and holes which must be completed by the cryptographic data. This could be sufficient to generate such a simple document.

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.

Remark 8. *ANSSI and EFA French Ministry chose Fix 1 to be implemented in the mid-term. Fix 2 seems to be the ultimate solution chosen by ANSSI and EFA French Ministry for a future version of the system. However we have no guarantee that it will be implemented if elections must be organized in 2023 because of time constraints. Unlike the Swiss Post system [18] about to start the production phase for Swiss elections, the EFA French Ministry does not make the choice of a continuous development of the e-voting solution (this can be explained by the number of elections to organize). Instead it organizes a public call for tenders every 4 years to choose the system that will be deployed for all the elections organized during this period. On this occasion, it defines a list of demands and closely works with the different companies to obtain a solution which matches all of them. Modification of the system afterwards seems too complex.*

We hope that this fix will be included in the new list of requirements of the next public call for tenders.

3.3 Attacking and Fixing Ballot Privacy

The second vulnerability can be additionally exploited to break ballot *confidentiality* (see Definition 1) of target voters, who are 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). Intuitively, the attack is as follows: the attacker gathers all the ballots but the target voter's one in the same ballot-box $\mathtt{u}_1$ and put the target voter's ballot alone (or with ballots whose the attacker knows the vote) in another one, $\mathtt{u}_2$ ($\mathtt{u}_1, \mathtt{u}_2$ are ballot-box identifiers $\mathsf{ballotBoxId}$). Because the ballot-boxes are individually tallied, the attacker will learn the plaintext vote of the target after the tally of $\mathtt{u}_2$. As we shall see, variants of this attack exist. For example, the attacker can use this attack to exercise an efficient coercitive power on as many voters as he wishes, in a fully remote way.[28] Indeed, the attacker can move all the target voters's votes to $\mathtt{u}_2$ and observe the result of the tally of $\mathtt{u}_2$. This way, the attacker can verify that all those voters did comply to his instructions to vote for a given candidate $v \in \mathcal{V}$.

Actually, the attack cannot be as simple because different checks are (or could be) done: the number of ballots in each ballot-box must match the number of signatures into the signing sheet[29], and a voter originally associated to $\mathtt{u}_2$ should not detect that his ballot has been moved in $\mathtt{u}_1$. We shall see that the attacker can mount our attack while satisfying those checks, thus completely evading detection. For sake of understandability, in the following we first present simplified variants of our main attack Attack_5[swapCs](breaks BallotPrivacy). Before presenting our fixes, we also describe another attack Attack_6[swapID](breaks BallotPrivacy) variant that exploits a different attack vector.

3.3.1 Attacking Ballot Privacy

In order to describe how the attacks work, we first recall how the signed seals are computed. We use the notations introduced in Section 2.4 and Figure 5 and, for the sake of simplicity, we omit the fields $\mathsf{errorCodes}$ and $\mathsf{pk_S}$ that are irrelevant to our attack. The ballot reference H and the digital signature σ are computed as follows:

$$\begin{aligned} H &= \mathsf{elecInfo} \| h \\ \mathsf{cSU} &= \mathsf{infoSU} \| \mathsf{sign}_{\mathsf{sk_S}}(\mathsf{hash}(\mathsf{infoSU})), \\ \mathsf{infoSU} &= \mathsf{elecInfoSU} \| \mathsf{ballotBoxId} \| hb, \end{aligned}$$

where:

- $\mathsf{elecInfo} = \mathsf{roundId} \| \mathsf{electionId}$ and $\mathsf{elecInfoSU} = \mathsf{roundId} \| \mathsf{electionId} \| \mathsf{electionName}$ are election meta-data that are equal to all ballot-boxes in the same election (constituency),
- $h = \mathsf{hash}(b \| \mathsf{elecInfo} \| \mathsf{ballotBoxId})$,
- $b = (c, \pi)$,
- $hb = \mathsf{hash}(b)$
- $\pi = \mathtt{zkp}$ (v is valid with context $\mathsf{electionId}, \mathtt{tokenId}$).

Attack_3[swapS](breaks BallotPrivacy), simplified variant under a voting Server attacker: In this first attack scenario, we assume a voting server attacker, *i.e.,* the voting server is compromised. We assume that the election (constituency) identifier $\mathtt{elec}$ contains at least two ballot-boxes (consulates) identifiers $\mathtt{u}_1$ and $\mathtt{u}_2$. Alice is the target voter whose vote will be revealed to the attacker. She is eligible to vote in consulate $\mathtt{u}_1$. Alice's cryptographic data are indexed by 1 while Bob's, expected to vote in consulate $\mathtt{u}_2$, by 2. The attack scenario is as follows:

- Alice votes as expected and produces and sends a ballot b_1.
- The voting server attacker receives b_1 but stores it in the ballot-box $\mathtt{u}_2$ (while Alice was actually eligible to vote in $\mathtt{u}_1$).

[28] 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.

[29] That said, we argue in Remark 5 why it is unlikely that voters check the signing sheet given how complex it is for them to access this sheet.

- In addition, the voting server attacker computes malicious receipts to make Alice get a valid receipt for a ballot in the wrong ballot-box, without Alice being able to detect the manipulation (for the sake of readability we use $\mathtt{elec}$ and $\mathtt{elec}'$ to respectively refer to $\mathsf{elecInfo}$ and $\mathsf{elecInfoSU}$ for the election and round in which Alice votes):

$$H_1^{s_1} = H_1^{s_3} = H_1^c = \mathtt{elec} \| \mathsf{hash}(b_1 \| \mathtt{elec} \| \mathtt{u}_1)$$

but

$$H_1^{s_2} = H_1^{s_4} = \mathtt{elec} \| \mathsf{hash}(b_1 \| \mathtt{elec} \| \mathtt{u}_2)$$
$$\text{and} \quad \mathsf{cSU}_1 = \mathtt{elec}' \| \mathtt{u}_1 \| \mathsf{hash}(b_1) \| \mathsf{sign}_{\mathsf{sk_S}}(\mathsf{hash}(\mathtt{elec}' \| \mathtt{u}_2 \| \mathsf{hash}(b_1)))$$

We can 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_1 cast in $\mathtt{u}_2$, which is important, as we shall see, to make all 3rd-party checks succeed and avoid detection. We will explain how after presenting Attack_3[swapS](breaks BallotPrivacy).

Note that this swap of ballot from a ballot-box to another one in the same election can be repeated at will. In particular, the voting server attacker can this way choose $\mathtt{u}_2$ such that there are very few eligible voters and therefore a few expected votes and then move all honest and legitimate ballots cast in $\mathtt{u}_2$, e.g. Bob's ballot, to another ballot-box, for example $\mathtt{u}_1$ in addition to moving Alice's ballot to $\mathtt{u}_2$. By revealing the tallied result for $\mathtt{u}_2$, the attacker learns how Alice has voted. We discuss later how is this practical by analyzing data from the real 2022 French legislative election and we shall see that such ballot-box $\mathtt{u}_2$ with very few eligible voters (*e.g.*, 23 registered voters and only one expressed vote) actually exist.

Remark 9. *In order to preserve a perfect correspondence between the number of ballots in* $\mathtt{u}_1$ *(respectively* $\mathtt{u}_2$*) and the corresponding items signing sheet, the voting server would need to balance out any swap, e.g., compensating a move of a ballot from* $\mathtt{u}_1$ *to* $\mathtt{u}_2$ *with a move of another ballot from* $\mathtt{u}_2$ *to* $\mathtt{u}_1$*. As we shall see with our ultimate attack* Attack_5*[swapCs](breaks BallotPrivacy), this can be easily done using compromised or colluding voters. The same applies here. This way, the attack becomes completely stealthily as explained in Remark 4.*

That said, we shall see in Remark 5 that it is unlikely that voters check the signing sheet.

Attack_4[swapC](breaks BallotPrivacy), simplified variant under a Channel attacker: In this scenario, we assume that the attacker only controls the (plaintext) communication channel. We assume Bob is any voter such that: he is an eligible voter in ballot-box (consulate) $\mathtt{u}_2$ and he casts a vote simultaneously when Alice casts her vote. Bob's cryptographic data are indexed by 2. The attack scenario is as follows:

- Alice votes as expected and produces and sends a ballot b_1.
- The channel attacker intercepts the ballot b_1 and replaces it by Bob's ballot, *i.e.*, b_2. In addition, he modifies the messages sent by the voting server to make Alice get a valid receipt for b_2, while she intended to cast b_1 instead:

$$H_1^{s_1} = H_1^{s_3} = H_1^c = \mathtt{elec} \| \mathsf{hash}(b_1 \| \mathtt{elec} \| \mathtt{u}_1)$$

but

$$H_1^{s_2} = H_1^{s_4} = \mathtt{elec} \| \mathsf{hash}(b_2 \| \mathtt{elec} \| \mathtt{u}_1)$$
$$\text{and} \quad \mathsf{cSU}_1 = \mathtt{elec}' \| \mathtt{u}_1 \| \mathsf{hash}(b_2) \| \mathsf{sign}_{\mathsf{sk_S}}(\mathsf{hash}(\mathtt{elec}' \| \mathtt{u}_1 \| \mathsf{hash}(b_2)))$$

We 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 ballot that will be added in $\mathtt{u}_1$ is Bob's ballot.

- similarly, the attacker intercepts b_2 and replaces it with Alice's ballot b_1. Again the attacker modifies the references and the seal sent to Bob replacing b_2 by b_1. This swap between Alice's and Bob's ballots maintain the correspondence between the number of elements in $\mathtt{u}_1$ (respectively $\mathtt{u}_2$) and the corresponding signing sheet.

Remark 10. *The scenario above describes a theoretical attack that might be difficult to exploit by an attacker who controls only the communication channels. Indeed, it requires that Alice and Bob vote at the same time. However, this scenario become much easier if either Bob colludes with the attacker or if the attacker controls the voting server. In the first case, the synchronicity is no longer necessary because Bob can create his ballot* b_2 *before Alice starts to vote. In the second case, the simpler variant* Attack_3*[swapS](breaks BallotPrivacy) becomes possible.*

Limitation. As previously mentioned, this attack scenario can be detected in specific contexts. For instance, if Bob is the unique voter in $\mathtt{u}_2$ and his plaintext vote is different from Alice's then he will be able to detect that something went wrong when having a look to the result of the election: Bob expects to see 100% of votes for his candidate v_2 but sees 100% of votes for Alice's candidate $v_1 \neq v_2$. We remedy this problem with our ultimate attack on ballot privacy we present next.

Attack$_5$[swapCs](breaks BallotPrivacy), completely stealthily variant under a channel attacker: We modify Attack$_4$[swapC](breaks BallotPrivacy) to prevent such detection. If we note $v_1, \dots, v_k$ the k voting options, let us assume that:

1. there are at least $k+1$ eligible voters in $\mathtt{u}_2$, say E is this number.
2. there exist k voters, $V_1, \dots, V_k$, known by the attacker and different from Alice 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, $V_{k+1}, \dots, V_{2k}$, different from Alice such that V_{k+i} is willing to vote for v_i in $\mathtt{u}_1$ for all $i \in \{1, \dots k\}$. In contrary to $V_1, \dots, V_k$, we do not assume that the attacker knows $V_{k+1}, \dots, V_{2k}$, 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, for the second round of the 2022 election, all of the ballot-boxes of the 11 elections got at least one vote for each of the candidates.
4. the attacker knows $E-1$ voters $V'_1, \dots, V'_{E-1}$ 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.[30]

We modify Attack$_4$[swapC](breaks BallotPrivacy), where two ballots are swapped between $\mathtt{u}_1$ and $\mathtt{u}_2$ (step 1. below), with the steps 2 and 3 below:

1. Alice's ballot is swapped with Bob's so that Alice's ballot b_1 is added to $\mathtt{u}_2$ and Bob's in $\mathtt{u}_1$;
2. 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 of $\mathtt{u}_2$ different from Alice and V_j for all $j \neq i$. These swaps ensure that during the tally, $\mathtt{u}_2$ contains at least one vote for each voting option $v_1, \dots, v_k$.
3. 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 of $\mathtt{u}_2$.

Thanks to the assumed existence of the voters $V_{k+1}, \dots, V_{2k}$, the result of $\mathtt{u}_1$ is not inconsistent in the point of view of Alice, who wanted to vote in $\mathtt{u}_1$, since at least one vote in $\mathtt{u}_1$ is the vote Alice intended to cast.

Impact. The impact of Attack$_3$[swapS](breaks BallotPrivacy) and Attack$_4$[swapC](breaks BallotPrivacy) is maximal when we assume that $\mathtt{u}_2$ contains only Bob's ballot, *i.e.,* all the other eligible voters abstain or use paper-based voting (since results are announced per ballot-box and per voting facility). In this case $\mathtt{u}_2$ will only contain Alice's ballot after the attack and the result of the tally will be Alice's plaintext vote. The privacy leak is immediate. We note that this 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). She might have decided to abstain if she knew that she would be the unique voter to vote.

In the 2022 French legislative election, we note that some consulates have received a unique ballot. For example, the consulate SVX-EKATERINBOURG which belongs to the 11th constituency received a unique (electronic) ballot during the first round among the 23 eligible voters. This ballot expressed a vote for the candidate Catya Martin[31]. For the sake of illustration, this unique ballot may not correspond to one of the 23 voters of the consulate SVX-EKATERINBOURG but could have been moved from one of the consulate in the same constituency which contains 98 853 eligible voters. The attacker could have chosen this target voter.

We do not claim that such an attack happened. We solely claim that a channel or a voting server attacker had the technical ability to perform such an attack without leaving any evidence we could exploit now to know if it has been exploited.

In practice, we note that many consulates received a small number of votes. For example MSQ-MINSK during the second round (6 electronic votes among 146 eligible voters) or SVX-EKATERINBOURG during the second round (6 electronic votes among 23 eligible voters). These two consulates belong to the same constituency

[30] 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.

[31] https://www.diplomatie.gouv.fr/IMG/pdf/leg_2022_t1_-_resultats_circo_11_-_6_juin_2022_cle4c19ff.pdf

having consulates with a large number of voters with potential targets such as SYD-SYDNEY which received 4049 electronic ballots during the second round[32]. Based on our attack scenario, we cannot be sure that these 2×6 votes, counted in the two first consulates, were not intended to be cast in the consulate of SYD-SYDNEY.

As shown with the more elaborate Attack_5[swapCs](breaks BallotPrivacy), the attacker is not limited to exploit consulates with a unique cast ballot. If a few voters have voted in $\mathtt{u}_2$, the attacker can swap all of them using either colluding voters or voters for which the attacker can guess how they vote for (the voters V_i'). The unique unknown vote in the result of the ballot-box $\mathtt{u}_2$ will be Alice's vote. Similarly, the attacker could decide to target several voters instead of just Alice and gather all their ballots into $\mathtt{u}_2$. This attack would give a power of coercion 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.,* ambassadors or diplomats. Those votes would normally be mixed with many other votes in their respective ballot-boxes, thus protecting their privacy. Attack_5[swapCs](breaks BallotPrivacy) shows that an attacker can isolate this population and learn their vote distribution.

Why is Attack_5[swapCs](breaks BallotPrivacy) completely stealthily? Let us review all checks that are performed. We already addressed the voting client and voting server checks in Remark 4. In particular, we recall that in the present attacks, all data received and processed by the voting server are genuine- and honest-looking, the voting server does not know how voters intended to vote.[33]

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 3rd-party, independent researchers who operate a server executing an open source program, named VVFE[34]. 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 $V_1, \dots, V_k$ in $\mathtt{u}_2$ and $V_{k+1}, \dots, V_{2k}$ 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, 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 remain 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 $\mathsf{ballotBoxId}$ (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.

[32] https://www.diplomatie.gouv.fr/IMG/pdf/resultats_leg_t2_-_circo_11_cle049798.pdf

[33] 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.

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

High level overview of the threat:
An attacker who compromises the communication channels (or even more so the voting server) can learn the plaintext vote of arbitrary target voters. The number of voters who can be targeted is immediately related to the number of consulates with a small number of votes cast. A large number of voters under coercion can also be impacted as all their ballots can be moved in order for the attacker to know if they did comply.
We stress that our attacks do not break the encryption scheme used to protect votes but rather exploit attacker-controlled leaks from the tally results.

Before discussing how to fix those main attacks, we present a last attack that we shall show can be fixed differently (with Fix 5 and Fix 4) but that assumes a stronger attacker, *i.e.,* voting server attacker. It relies on the same mechanism, moving around ballots from one ballot-box to another one, but it does not directly exploit the vulnerability V_1.

Attack$_6$[swapID](breaks BallotPrivacy), swap ballotBoxId by a voting server attacker: Note that ballotBoxId is sent by the voting server to the voting client, right after authentication in the HTML document `generic_vote.htm`, along with `tokenId`. Therefore, a voting server attacker can modify the honest, legit ballotBoxId value (corresponding to the consulate where the voter is eligible) to any attacker-chosen ballotBoxId value and accepts the ballot and ballot reference the voting client will compute using the malicious ballotBoxId. This other attack vector can be exploited to mount the same attack scenario as Attack$_5$[swapCs](breaks BallotPrivacy). As opposed to the latter, the present attack makes the voting client computes a ballot and a ballot reference for the wrong ballotBoxId, but because its value is never displayed to the voter and because the voting client cannot know what is the legitimate value of ballotBoxId, the attack remains stealthily.

Remark 11. *One may argue that this attack is invalid under the assumption of an honest 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 4). If a piece of data that is received, send, 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 election key, 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 this attack in scope.*

3.3.2 Fixing Ballot Privacy

We propose three solutions to prevent the privacy attacks we found. Fix 3 is the simplest and has already been implemented by Voxaly Docaposte thanks to our findings.

Note that fixing the first vulnerability V_1 with Fix 1 or Fix 2 would already prevent our first three privacy attacks. However, our last privacy attack Attack$_6$[swapID](breaks BallotPrivacy) remains valid under a stronger voting server attacker and requires both Fix 4 and either Fix 3 or Fix 5. Moreover, we believe the second vulnerability we found V_2 should be addressed independently with Fix 4 as it introduces a weakness in the protocol, even though we do not think it could be exploited on its own by a channel attacker.

Fix 3: rely on the 3rd-party to detect attacks. In this first solution, the 3rd-party party would look-up the reference or the hash value regarding to a specific ballot-box (consulate) identifier. This piece of information is already available in plaintext in the seal cSU and can be verified to be consistent with the one in the ballot reference H. If Fix 4 was also implemented, the 3rd-party should also verify the ballot-box (consulate) identifier is the same in the ZKP of the ballot and in H (and in cSU). As an alternative, the 3rd-party could display to the voter the consulate in which his/her ballot has been cast and counted. This solution is easy to implement but makes the voter's journey and tasks more complex: the voter must do an extra check, either looking at the reference/seal (that may be complex because a the use of `base64` encoding of the data) or verifying an extra information displayed by the 3rd-party.

However, this solution is of interest to show that a 3rd-party, *i.e.,* an external auditor, can be useful not only to ensure verifiability, but also vote privacy. This new paradigm is interesting even if it applies only to

voters who verify. However, we prefer the next Fix 4 (in conjunction with Fix 5) that is more secure as it cryptographically binds a ballot with a ballot-box and thus does not rely on assumptions about voters' actions.

Fix 4: add the consulate identifier `u` in the context of the ZKP. A more straightforward solution is to add the ballot-box (consulate) identifier `u` in the context of the ZKP. This simple modification cryptographically prevents any swap of ballots between different ballot-boxes. This way, the claim written in the partial specification and mentioned in Section 3.1 becomes correct.

Remark 12. *This fix is in line with a cryptographic good practice which consists in adding all the public information available at the creation of the proof in its context. More on this in Section 4.*

Fix 5: make the voting client display the consulate identifier `u` during the voting process. Even if Fix 4 was implemented, ballot privacy would still be at risk under a stronger threat model. Indeed, a voting server attacker could still exploit Attack_6[swapID](breaks BallotPrivacy) so that the ballots honestly generated by the voting client can be built on the wrong, attacker-chosen ballotBoxId. Therefore, we recommend coupling Fix 4 with displaying the ballotBoxId (or better the readable name of the corresponding consulate) used for computing the ballot. This way, the voter could notice it is computing and casting a ballot in the wrong consulate. Attack_6[swapID](breaks BallotPrivacy) is thus made detectable and voters can stop the process of casting their ballot when under attack.

Note that Attack_6[swapID](breaks BallotPrivacy) can also be fixed by implementing Fix 3 and Fix 4 in that the voters who will verify their vote using 3rd-party will be able to detect that the voting server cheated. Since this attack detection is only possible for voters who check their ballots, we prefer the combination of Fix 4 and Fix 5, even though it requires to modify the UI of the voting client to implement Fix 5.

3.4 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 French Ministry, ANSSI, and Voxaly Docaposte.

3.4.1 Malleability of the ZKP

ZKP are complex cryptographic primitives that must be carefully used. In particular, as mentioned in Remark 12, 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 [10], 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 $b' = (c_2, \mathsf{zkp}_2, c_1, \mathsf{zkp}_1, zkp_{all})$ is thus a valid ballot. This malleability weakness could be exploited by a channel attacker in combination with the vulnerability V_1 to modify the choice of a voter (if b codes for "yes" then b' will code for *no*) even though the attack Attack_2[Replace](breaks Indiv.Verif) is more efficient and works under the same threat model. In [10], 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 Section 3.4.2.

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 [9], 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 [9], 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 [9] that is as easy as implement as Fix 4, which has already been implemented by the vendor.

3.4.2 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 [10] and their concrete impacts have been recently studied by Mestel *et. al.* [15].

The attack is quite simple: the attacker replays Alice's ballot to amplify its impact on the result and thus learn 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.* [15] 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[35]. 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.

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 13. *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.*

3.4.3 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

[35] 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 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 French 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 French 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 sets 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 French 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 Section 4.2.1.

3.4.4 (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 eligible 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 by 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, he 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 he casts, 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 French 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 French 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.

3.4.5 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 [8, 6, 13]), 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 [3]). 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 11, 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 web site 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.[36] Moreover, it puts an important trust assumption on the authority in charge of the distribution (public government, Google, Apple, etc.).

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

- 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[37]. Ideally, the code would be distributed in the form of a Single-page Application (SPA)[38] 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 French Ministry and ANSSI to include them in the requirement lists for the next public call for tenders.

4 Lessons Learned

We presented our security analysis and its outcomes. 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.

4.1 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 2.3.)

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 2).

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.

[37] Continuing Remark 11, 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 3 (or, better, of Fix 5).

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

4.2 Reflect the Use Case Specificities in the Cryptography

4.2.1 Operational and Lawful Constraints

We have shown in Section 3.4.3 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 [12] for specific quorum rules. The system allows to define mandatory authorities whose participation is necessary to decrypt. Regarding the quorum rules presented in Section 3.4.3, 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 14. *Perfectly encoding the quorum rules in the cryptography would not seem appealing for the EFA French 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 French 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 result 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.

4.2.2 Put the Whole Public Context in ZKP, Again

We have shown in Section 3.1 that a critical contextual information (ballotBoxId) was missing from the ballot ZKP. We also explained in Section 3.4.1 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 [9, 10, 14]. 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.

4.3 Simpler is Better

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

4.3.1 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 3rd-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[39], 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).

Signing sheet. Another important aspect is the access for the voters to the signing sheet discussed in Remark 5. 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, he may be muddled by the fact that he can do a first check during the voting phase (where only the validity of the signature of the seal cSU is checked) but he 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)).

[39] 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

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.

4.3.2 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.

4.4 Transparency and Specification

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

4.4.1 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 French Ministry) provided such a specification themselves. Ideally, it could be made a lawful requirement as it is the case in Switzerland (see Remark 15), 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 15. *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)*[40]*.*

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.

4.4.2 Need of 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 Section 4.1, a complete specification of the voting client is lacking. As we have seen in Section 2.3, 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 French 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 15). In the French context, the helpful discussions we had with the EFA French 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:

- 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 Test 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 [7] against the Swiss Post e-voting system. Whether variants of those attacks also break other e-voting systems remains an open question.

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

- 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 Section 4.2). 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.

5 Conclusion

We conduct a comprehensive security analysis of the FLEP for two central goals of e-voting protocols: verifiability and ballot privacy. Analyzing the French legal framework and relevant recommendations, we define a precise threat model specification that we argue is in line with the French legal framework and which is also supported by the literature. Due to a lack of specification of the protocol and most notably of the voting client, we had to reverse engineer the protocol to build a complete specification of the FLEP. To do so, we notably had to overcome the obfuscation of the voting client JavaScript code and cross-reference multiple code bases and sources. We thus contribute the first complete public specification of the FLEP.

We then report on the results of our analysis of the FLEP we could conduct thanks to the above specification. We report on two major vulnerabilities: one implementation-level flaw in the voting client which defeats the purpose of ballot references of achieving verifiability and one design-level flaw in the ballot construction for which we show a contextual information is missing from the ZKP. We then describe and discuss 6 concrete attacks exploiting those two main vulnerabilities and other flaws that break verifiability and ballot privacy in the FLEP. Those attacks, the threat model under which they are possible, their impact, and how they will be fixed are summarized in Table 3. The first most important attack stealthily breaks verifiability and the election integrity under a channel attacker, *i.e.,* an attacker who is able to compromise the secure channel which is a weaker attacker than a compromised voting server. The second most important attack stealthily breaks ballot privacy of target voter(s) under a channel attacker, that is a channel attacker can learn how some targeted voters voted. The detailed description of the attacks, their threat model assumptions, and their impact, are discussed in Section 3. We responsibly disclosed those attacks to the relevant stakeholders, who acknowledged our specification and attacks. We respected a 3 month embargo period (starting months after the end of the actual election) to discuss 5 different fixes we suggested to them to prevent our attacks and secure the FLEP in the future. 3 fixes, which resolve the most critical threats, are expected to be deployed soon and are summarized in the same table.

Moving forward. Our findings are interesting beyond the specific case of the FLEP and we draw more general lessons and recommendations from this experience.

Indeed, the FLEP case study is especially relevant as it shows how things can go bad with the excessively challenging real-world deployment at scale of e-voting solutions. To do so, many aspects that are very often omitted in the academia have to be taken into account such as the legal requirements and recommendations that come with political elections, compliance to a competitive public call for tenders, etc. This could explain why, despite being derived from the state-of-the-art academic protocol Belenios [8], the FLEP had to be instantiated and modified to comply with those additional constraints and some of those modifications introduced the vulnerabilities and attacks we found, which were inexistent in Belenios. From this FLEP experience, we discuss interesting factors that could explain why is this so and we draw more general lessons and recommendations that are relevant both for academic researchers but also for election organizers and vendors to better secure political elections in the future.

First, it appears that organizers 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. For example, we have shown the critical importance of the voting server attacker model.

Second, the academic solutions can do better at offering enough modularity and generality to adapt to real-world constraints. For example, if Belenios included a multi ballot-boxes (for the same election) option, the FLEP would not have to add ballot-box identifiers while forgetting to include it in the ZKP (second vulnerability). Similarly, if Belenios did offer PDF receipts then the FLEP would not have to introduce new receipts while forgetting to verify that they are all consistent (first vulnerability).

We also explain why complex protocol and notably voter's journey and verification tasks are detrimental to the overall protocol security. In the case of FLEP, we show that those could have been simplified a lot and would have make the first vulnerability much more obvious.

Finally, because we all know that designing a secure system is a difficult task, we would like to promote transparency and advocate for more public scrutiny from different communities (academic researchers, hackers, etc..). This could be incentivized with public intrusion test and bug bounty programs. Moreover, while great care has been put in securing the voting server against internal and external attacks, which seems to usually be a strong expertise of large companies, auditors, and certification authorities, the voting client has seemingly received much less attention. Yet, our analysis sheds light on the critical role of the voting client provided to the voters that must be considered a primary analysis target. This piece of code is indeed the keystone for the verifiability of the system. We advocate for making the voting client fully public and amenable to public scrutiny and external audits (with a public specification and open-sourced code, ideally FLOSS).

In conclusion, FLEP 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 1.1 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. There is hope though. We had very insightful discussions with the different stakeholders and we strongly believe more of such communication between academia and e-voting practitioners is for the better. We are confident that the FLEP stakeholders will take our recommendations into account for the design, the choice, and the analysis of the next e-voting system to be used in France. We expect a public call for tenders with a clearer threat model and security objectives, a better transparency, and hope for concrete solutions proposed by the academic community to solve open questions raised in this work.

References

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

[2] 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.

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

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

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

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

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

[14] Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. Addendum to how not to prove your election outcome, 2019.

[15] David Mestel, Johannes Mueller, and Pascal Reisert. How efficient are replay attacks against vote privacy? a formal quantitative analysis. Cryptology ePrint Archive, Paper 2022/743, 2022. https://eprint.iacr.org/2022/743.

[16] Journal Officiel. Délibération n° 2019-053 du 25 avril 2019, 2019. Document téléchargé et accessible le 28 juin 2022 via le lien https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000038661239.

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

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

[19] Voxaly. Voxaly, partenaire du Ministère de l'Europe et des Affaires Étrangères (MEAE), éléments techniques, 2022. Document téléchargé et accessible le 28 juin 2022 via le lien https://w8t9w2j6.stackpathcdn.com/wp-content/uploads/VOXALY_LEG2022_Verifiabilite_Specifications.pdf obtenu à partir de la page https://www.voxaly.com/vote-par-internet-pour-les-francais-de-letranger-dans-le-cadre-des-elections-legislatives-2022/.

A 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 [11].

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 [11].

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.