libharpia: a New Cryptographic Library for Brazilian Elections

Rodrigo Pacheco[1], Douglas Braga[1], Iago Passos[1], Thiago Araújo[1], Vinícius Lagrota[1], Murilo Coutinho[1]

[1]Research and Development Center for Communication Security (CEPESC)

Abstract. *The Research and Development Center for Communication Security (CEPESC) has a long partnership history with the Brazilian Superior Electoral Court to improve the security of the Brazilian election system. Among all the contributions from CEPESC, probably the most important is a cryptographic library used in some critical moments during the election. In an effort to improve transparency and auditability of the solution, we present the new cryptographic library developed at CEPESC, named* `libharpia`*. Its main design goal is to allow transparency and readability while substantially increasing security. One of the main advances is the use of post-quantum cryptography, implemented through secure hybrid protocols that mix current cryptographic standards (specifically elliptic curves) with new cryptographic primitives based on Lattices, believed to be secure against quantum computers.*

1. Introduction

The Brazilian electronic voting machine, or simply "Urna Eletrônica" (UE), was developed by the Brazilian Superior Electoral Court (Tribunal Superior Eleitoral - TSE) and is part of the Brazilian electoral process since 1996. Along these years we have seen great progress in the system and in the process itself. In the beginning, the UE was based on personal computers, implementing a VirtuOS and Windows CE, and was designed by third parties. From 2005 onwards, the development of the most critical parts of the systems has been developed by TSE's engineers, including a new operating system based on Linux deployed in 2008 [Monteiro et al. 2019]. In 2009, the security of the system was greatly improved by the adoption of a hardware trusted computing base (TCB) for direct recording electronic voting architecture, T-DRE in short. This device is capable of performing several important cryptographic functionalities such as key storage, random number generation, digital signatures and the authentication of critical softwares (e.g., BIOS, bootloader, Linux kernel) [Gallo et al. 2010]. Another great advance was the use of biometric data to uniquely identify the voters, a process that started in 2008 and now reach over 80% of the electorate. More recently, TSE announced that new voting machines are going to use digital certificates from Brazilian national PKI (ICP-Brasil) [TSE 2021].

All these initiatives showcase TSE's efforts into making Brazilian elections secure and trustworthy. Nevertheless, from the beginning, the Brazilian voting system has been criticized in terms of its transparency and auditability. For example, in 2002, a report from the Brazilian Computer Society (SBC) presented a complete analysis of the election system. Although no signs of fraud or big security problems were found, the authors appointed a lack transparency and auditability as

a concern [van de Graaf and Custódio 2002]. Another report with similar conclusions [Brunazo Filho et al. 2015] was published after a very tight election in 2014.

Along the years, TSE has been working to improve these limitations. Starting in 2009, one important step in this direction was the organization of several hacking challenges, the so called "Teste Público de Segurança" (TPS), where external and independent researchers can examine the security mechanisms implemented within the system, find vulnerabilities and provide suggestions for improvement (see [Aranha et al. 2019] for a review of the problems detected in these tests). We note that the vulnerabilities found are always patched by TSE and the fixes presented to researchers. In addition to the TPS, TSE has progressively been opening the system by open-sourcing parts of the solution [Alessandre 2021] and through academic publications [Coimbra et al. 2017, Monteiro et al. 2019].

However, there is still room for improvement. For example, one common criticism is the participation of the Research and Development Center for Communication Security (CEPESC) in providing cryptographic libraries for TSE. Indeed, a report published in 2015 said that "TSE accepts in the electoral system the existence of code and services of authentication provided by CEPESC, constituting a critical point of failure that can influence almost all routines in the electoral system" [Brunazo Filho et al. 2015]. The criticism was valid, since up to that point, the source code developed by CEPESC was not known by the community and was not a part of the TPS.

Created in 1982, CEPESC has four decades of experience, research and development in cryptology, there is no other institution in Brazil with such an experience in the area. CEPESC's primary mission is to develop secure solutions to protect the most critical information of the Brazilian government. CEPESC has a long partnership history with TSE, being an important part of the electronic election system since its inception in 1996. During this period, CEPESC helped TSE in several critical activities, such as: (i) evaluating the security of critical software and hardware components developed by third parties; (ii) testing the quality of the random number generators of the UE and its peripherals; (iii) helping TSE to develop protocols and evaluating operational security; (iv) developing cryptographic libraries and algorithms.

Traditionally, CEPESC's solutions were treated as proprietary and their source code were never disclosed to the general public. This reality is changing in order to improve transparency and auditability of the election system, both key aspects of democracy. This work is a further step in this direction, as the organizational agreement with TSE already establishes the necessity of opening the design choices and possibly the source code itself. Therefore, in this work, we describe the new version of the cryptographic library developed by CEPESC, called `libharpia`. One of the goals of this work is to achieve a higher social trust in the primitives and protocols implemented.

The main advantage of `libharpia` is the use of post-quantum cryptography implemented through secure hybrid protocols that mix current standards (based on elliptic curves) with new cryptographic primitives (based on lattices). More precisely, we use the algorithms Kyber and Dilithium [Ducas et al. 2018, Bos et al. 2018] to deliver a Key Encapsulation Mechanism (KEM) and digital signatures, respectively. These algorithms were recently announced among the winners in the post-quantum cryptography standard-

ization process organized by NIST [NIST 2022]. To the best of our knowledge, this will make Brazilian elections the first in the world using post-quantum cryptography and the first system in general using this kind of cryptography in any Brazilian public institution.

This work is organized as follows: in Section 2, we provide a description of the goals and scope of `libharpia`. In Sections 3 and 4, we present the main choices in terms of implementation and cryptographic primitives, respectively. Then, in Section 5 we present the API of `libharpia`. Also, in Section 6, we go into more details over the cryptographic protocols implemented, providing justification for their choices and security. Finally, in Section 7 we present the conclusions.

2. Goals and Scope

`libharpia` is used by TSE in the following situations:

1. The TSE prepares over 500 000 UE machines for the election. To do so, a media containing all software and data necessary is created and used to inject them into the UE. To make sure that an adversary cannot create a fake media, all binaries are signed using `libharpia` and TSE's private key to prove their authenticity.
2. During the election, voters are authenticated using their biometric information. To make sure that this private information is protected, `libharpia` is used to encrypt all the data.
3. When all voters from a particular section finish voting, the UE emits a ballot box ("Boletim de Urna"), that is a summary of the results of a specific machine. Then, the UE encrypts and signs the ballot box using `libharpia`. This step protects the ballot box from tampering during transmission.
4. TSE's tally server receives the encrypted ballot box from every single UE machine across the country. Then, it verifies the digital signature and decrypts the ballot box to access the results and add them to conclude the election.

It is important to note that `libharpia` is not the only means of cryptographic protection in the electoral system. In fact, TSE uses several layers of security to minimize any risks of errors or vulnerabilities in the system. Additionally, we stress that CEPESC does not have access to the core implementation of TSE or any of their systems, and that `libharpia` is just a dry cryptographic API that could be used in any other system. Therefore, it is TSE's responsibility to use `libharpia` correctly.

3. Library Design

In order to facilitate the inclusion of new algorithms and interchangeability of the designated algorithms, the architecture of the library has to be modular. This is achieved by working with a layered design based on libsodium [Denis 2013] and NaCl [Bernstein et al. 2012], where in the lowest layer we have the implementations of all the primitives of the library, AVX2, SSE2, reference implementations, and any other applicable instruction extension set. A middle layer that standardizes cryptographic operations, by doing what is common to, for instance, any stream cipher independent of what implementation one is using. And a top layer that is responsible for interfacing with the user with a higher level of abstraction. Providing support to these three layers, there is a set of utilities, composed of constant time implementations of commonly used functions,

mainly buffer comparison and a centralized entropy source. This is the `core`, and it is highlighted in blue in Figure 1.

A fourth layer is above the `core`, which is responsible for the actual user interface TSE uses. This layer exists solely to fulfill API retrocompatibility and facilitate the integration of our library in TSE's systems. This layer also has some utilities related to legacy functionalities such as an ASN.1 and AES implementations used in a key encryption protocol. In the future, a new API will be negotiated with TSE, and this fourth layer will not be necessary, which will improve maintainability.

The modularity comes from the interfaces between these layers. When a new primitive is added, by complying with the interface, turning that primitive into the designated primitive is a matter of changing pointers. Despite not much more code being necessary to allow a change of primitives during runtime, `libharpia` defines its primitives when compiling, in order to comply with our strict versioning premise.

The interface between bottom and middle layer allows written code in middle layer to abstract which implementation is being used. For instance, a stream cipher needs to present a function pointer that takes a nonce, a key and a length, and outputs a keystream. The middle layer will regard the function pointer presented, and not which implementation provided it (AVX2 or reference, for example). In the middle layer we adjust the primitives, to conform with the interface between top and middle layer, which is to say that enough information about the primitive needs to be passed to the top layer so that sanitizing user inputs can be done on the top layer, abstracting the primitive.

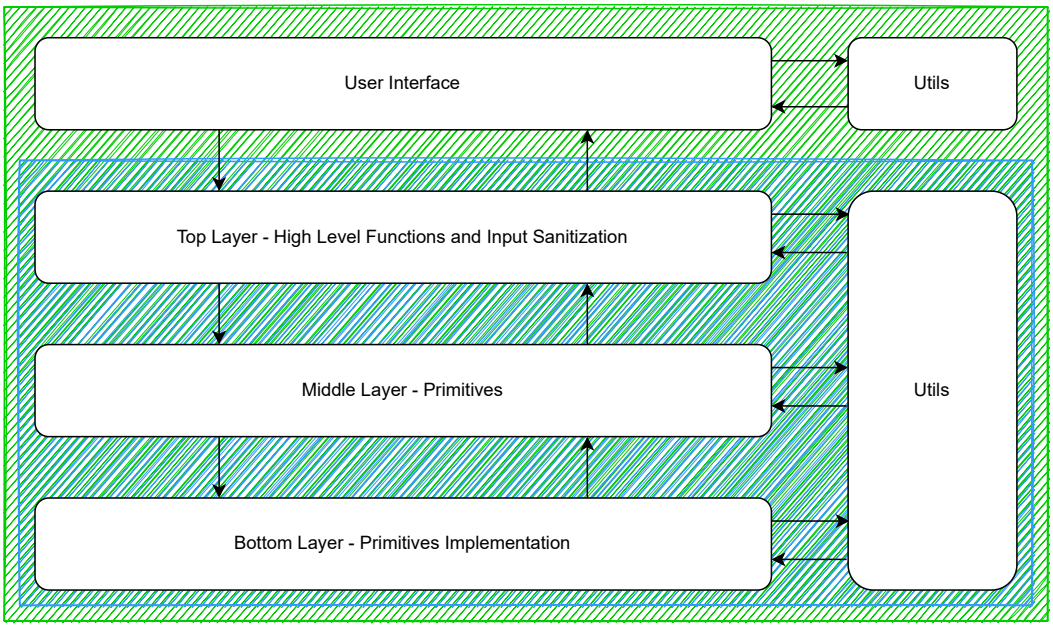

Figure 1. libharpia's architecture.

Based on this discussion, the main difference of this library when compared to the libraries we based our work on is the use of hybrid public-key protocols, including post-quantum algorithms. Another important difference is the nonce generation - `libharpia` generates a random nonce for each encryption while both libsodium and NaCl leaves the nonce management to the user. This choice protects the overall solution against common

implementation errors from the developers, such as nonce reuse.

Next, we describe a series of security principles adopted in the design and implementation of `libharpia`.

3.1. No Heap allocation and memory management

Abusing the heap is a very common vector of attack on secure systems. Improper handling of crashes, heap overflows, all contribute to nullifying security [Novark and Berger 2010], specially for modern solutions that involve co-hosting, or simply sharing resources with an adversary process. Nevertheless, this risk can be mitigated through careful implementation. In the development of a cryptographic library, it is more easily done by avoiding the use of heap allocation in the library, and relying on the stack.

Furthermore, to mitigate the impact of other types of side-channel memory attacks, we implement a stack memory management to ensure that every memory address that contained secret information (e.g., keys, plaintexts, cipher states) is erased as soon as it reaches its final use. This way, we seek to minimize the lifespan of sensible data in the machine.

3.2. No data flow from secrets to load addresses or branch conditions

Since 2005, it has been known that if any secret influence load addresses, a viable vector of attack is created by timing an unprivileged process in the same machine [Bernstein 2005]. Once these properties influence performance, different keys impact the adversarial process differently. Thus, by measuring this impact, cryptographic keys can be discovered almost instantly in implementations that use, for example, fast lookup tables [Pointcheval 2006, Tromer et al. 2010]. We avoid any implementations where secrets are suspected or known to affect performance.

Another source of timing side channel attacks is secrets being used as branch conditions. Admittedly, the choice of primitives helps here, since there are primitives that by design favor constant time. But more importantly, the implementations used are checked using `valgrind` [Langley 2010].

3.3. Centralizing randomness - cryptographic secure random number generator

Centralizing randomness gives us safety through maintainability. By focusing our attentions on a single source of randomness, we can ensure that any fix or improvement made on it is reflected across all the library. No primitive in our library gets entropy through any other means. By having changes in the source code follow a strict suite of tests on the quality of the random number generator, this culminates in a safer library.

3.4. Nothing is decrypted unless it first survives authentication

Encryption without authentication is not offered. No data is decrypted if authentication fails. Additionally, authentication verification occurs in constant time.

3.5. Strict versioning

As the library is designed to work only in the Brazilian Elections system, it will be used in a very closed environment. The library is designed to be able to communicate only

with others in the same version, so when a new version is released, all components of the system must be upgraded to this version.

This approach avoids downgrade attacks, where an attacker can exploit security flaws in older versions even if the system is running an up-to-date version [Alashwali and Rasmussen 2018]. Although no backwards compatibility is too rigid for decentralized environments, this strategy is reasonable to implement in our scope as the library distribution to the components is controlled by a centralized entity, the TSE.

3.6. Key generation

All persistent (non-ephemeral) assymetric key-pairs used in the library are generated using an external program to the library. This program communicates with a cryptographic token developed by CEPESC, the PCPv2 (Portable Cryptographic Platform). PCPv2 is an USB device with secure storage and processing capabilities and provides, among other cryptographic functionalities, a physical True Random Number Generator.

A sequence of random numbers is provided by PCPv2's TRNG and combined with the machine entropy pool as a seed to the key generation program's PRNG. As the random number generation for keys is a critical part of every cryptographic implementation and weak RNGs are the primary source of numerous practical attacks on secure systems, we opted to have a conservative approach using TRNGs. Furthermore, the double-source of entropy approach is able to mitigate risks of entropy pool poisoning.

4. Cryptographic Primitives

One of the main aspects of `libharpia` is that its security is founded on the security of the primitives and protocols used. A rule of thumb for choosing good cryptographic primitives was to avoid secret and untrusted ones. Additionally, we focused on maximizing performance in software and minimizing memory usage. As we aim to 256 bits security for the standard operations of the library, some primitives were also not considered.

We chose the combination Chacha20-Poly1305 for authenticated encryption with associated data, as it is referred in [Nir and Langley 2018]. The first question that comes in mind is "Why not use AES?". The simple answer is that Chacha performs better than AES on software, and in most CPUs without hardware acceleration, with greater security margin. Chacha20 is also intrinsically time safe. Another aspect of concern is the higher complexity of AES, which still is subject to hidden relations, for instance, recently some surprising properties were found for its key schedule [Leurent and Pernot 2021]. Poly1305 is a message authentication code and its security is based on the security of the underlying algorithm, which means that if one could break Poly1305 than one would also break Chacha20. We also note that CEPESC's researchers extensively studied the security of ChaCha [Coutinho and Neto 2021].

The standard choice for hash function in `libharpia` is Blake2 [Saarinen and Aumasson 2015], which is an improved version of Blake, a finalist of the SHA3 competition. Blake2 has seen widespread use across cryptographic libraries (e.g., OpenSSL, WolfSSL and Sodium), and it is also used as a RNG for the Linux Kernel. It is faster than MD5, SHA1, SHA2 and SHA3 while been as safe as SHA3.

`libharpia` really stands out amongst other libraries in its use of post-quantum cryptography through hybrid protocols. In the classical part, we use Curve448 for

key exchange and its Edwards counterpart Ed448 for digital signature [Hamburg 2015]. Curve448 uses Solina's prime $p = 2^{448} - 2^{224} - 1$ which has the special format $p = 2^{2\phi} - 2^{\phi} - 1$, $\phi = 224$, resembling the golden ratio, from where it received the *Goldilocks* nickname, and allows for fast Karatsuba multiplication. Moreover, the curve has fast and friendly implementations for 32 and 64 bits platforms. The curve is considered safe under all the parameter from `https://safecurves.cr.yp.to/` and it is adopted by the new TLS 1.3. Although its expected security is 224 bits, bellow our aim, it is high enough for our purposes, since the hybrid protocols ensure that both primitives have to be compromised in any successful attack, and our choices for post-quantum primitives are believed to have at least 256 security.

With the post-quantum cryptography standardization in its final stages, we can clearly see the prevalence of lattice based cryptography. As a matter of fact, only two out of seven submissions are not lattice based, and the one base on multivariate equations was recently completely broken [Beullens 2022]. Our choice was to use the CRYSTALS family, that is, Dilithium for digital signature and Kyber for key encapsulation. Both algorithms rely on the Learning With Errors problem, which were proposed in [Regev 2005].

The CRYSTALS constructions were built from module lattices, that is, lattices drawn from more structured spaces, in this case, subspace of modules. The digital signature algorithm Dilithium [Ducas et al. 2018] is based on the Module-LWE problem, where the error is taken uniformly over the correspondent ring. Its signature scheme is based on the Fiat-Shamir With Aborts proposed by Lyubashevsky [Lyubashevsky 2009] and has several types of encoding in order to reduce its key and signature sizes. It performs better and has smaller public key plus signature size compared to all lattices submissions. Kyber [Bos et al. 2018] is a key encapsulation algorithm also based on the Module-LWE problem. It achieves indistinguishability under adaptive chosen ciphertext attack using the Fujisaki-Okamoto transform over a public key encryption algorithm.

As a final note, it is important to mention that recently the stated security of all LWE finalist were reduced accordingly to the Matzov center [MATZOV 2022]. In essence, the reduction were due improvement in the algorithms from the known dual lattice attack on LWE. In our view, this does not compromise the whole scheme because a change of parameters (one of the main advantages of considering modules) could mitigate the reduction. That said, as the NIST competition also decided for the CRYSTALS family, there is no current reason to change our original choice.

5. API

In this section, we present some of the functions provided by `libharpia`. The functions that are not listed here are simple variations of the presented ones and exist mainly for backward compatibility. We refer to Section 6 to more details about the protocols implemented in these functions.

The library's API is a requirement defined by TSE. Because of that, some of the elements of the API are present only to fulfill retrocompatibility requirements. Let's look at encryption as an example. We start with a initialization function:

```
rv = init_encryption(k,NULL,0,ct,ctl,pk,NULL);
```

As the name implies, `init_encryption` is responsible for any preparation necessary

for encryption, more precisely, `init_encryption` executes a hybrid key exchange using the keys that were generated preemptively, as explained in Section 3.6. The parameter `k` is a pointer to a memory area that will receive the symmetric key from the hybrid key exchange. `ct`, `ctl` and `pk` are pointers to, respectively, the encapsulated symmetric key, its length and the public key. Both `NULL` parameters are innocuous and the `0` could be any 64 bit unsigned int, which are present only to achieve retrocompatibility with the old API calls. Any `rv` different than `0` indicates an error. These error are designed to not give any critical information about the failure, but rather indicate what element was at fault, for example, the ASN.1 encoding or incompatible library versions. The `rv` behaves the same for all the functions.

Next, the encryption function is quite straight forward:

```
rv = encrypt(k,p,pl,c,cl,NULL);
```

Giving as input the symmetric key `k`, that one have already obtained using `init_encryption`, the plaintext `p` and its length `pl`, one will get the ciphertext on pointer `c`. The ciphertext has the same size as the plaintext plus 44 bytes and its length has to be inputted in `cl`, as a sanity check. Those 44 bytes are used for authentication with associated data and are composed of 28 bytes of metadata such as the library version, plaintext length and nonce, and the remainder 16 bytes for the authentication tag. Algorithm 3, presented in Section 6.4, explains the Authenticated Encryption scheme And like all other relevant length values, it is provided as a macro in our headers. Again, we note that `NULL` is only used for retrocompatibility.

Signing has a much simpler interface. Taking a buffer and a private key, and outputting the signature:

```
rv = sign_buffer(b,bl,s,sl,pk);
```

where `b` is a pointer to a buffer to be signed, `bl` is its length, `s` is a pointer where the signature will be outputted, `sl` is the length of the signature, an input serving as a sanity check, and `pk` is a pointer to the private key.

Naturally, we have `init_decryption`, `decrypt` and `verify_buffer`, which are analogous to the functions presented so far. We also provide a key derivation function:

```
rv = derive_key(sk,dk,salt,saltl,info,infol,NULL);
```

where `sk` is a pointer to the symmetric key, `dk` is a pointer to the memory address where the derived key will be outputted, `salt` is a pointer to the salt used in the key derivation process, `saltl` is a pointer to its length which should be a minimum of 128 bits, `info` is a pointer to any additional information to the key derivation (e.g., voter ID), `infol` is the length of all the info added.

6. Cryptographic Protocols

In this section, we present four cryptographic protocols implemented in our library. Each of those protocols is directly linked to functions provided by the API.

The signature and the key exchange are public key cryptography protocols and the classical methods for those protocols (RSA, elliptic curves) are being threatened with the

possibility of quantum computing and Shor's algorithm [Roetteler et al. 2017]. Therefore, we decided to use hybrid public key protocols, combining classical elliptic curves with candidates algorithms approved for the third and final round of NIST post-quantum cryptography standardization process [Alagic et al. 2020]. The rationale behind the hybrid approach is the conservative one: the protocols design aim to have at least the security of the most secure of both primitives in the classical and the post-quantum setting. In this way, we are protected in the non-quantum scenario with the well-known security of the elliptic curves and possibly in the quantum scenario with algorithms that relies on problems believed to be hard-to-solve in quantum computers.

The third protocol is a symmetric-key derivation protocol that derives a new symmetric key from a previous one using some associated data, used in the generation of the encryption keys of individual data of the voters. And, finally, the fourth protocol is the symmetric-key authentication encryption/decryption scheme, that must be used with keys derived from the key-exchange protocol or from the symmetric-key derivation protocol.

6.1. Hybrid Signature

The hybrid signature protocol is a simple signature protocol that combines elliptic curve signature using Ed448 and post-quantum signature using the lattice algorithm Dilithium5 presented in Section 4. The hybrid signature $S_m^{hy} = Sign^{ec}(m, SK_s^{ec})||Sign^{pq}(m, SK_s^{pq})$ of a message m is a concatenation of the elliptic curve signature of m and the post-quantum signature of the same message using the sender's elliptic curve and post-quantum secret keys SK_s^{ec},SK_s^{pq}, respectively.

To verify the signature S_m^{hy} of a message m, the verifier splits the signature in S_m^{ec}, S_m^{pq} and verify each part separately with the sender's public keys PK_s^{ec}, PK_s^{pq} such that the signature is only valid if both signatures are valid for m. For a forgery in this protocol, the adversary needs to forge both signatures, so this protocol is at least as secure as the most secure of both primitives.

The verification is always done for both signatures even if the first one already fails. This strategy is used to achieve constant-time verification and therefore avoid leaking which part of the hybrid signature failed the verification.

6.2. Hybrid Key Exchange

The hybrid key-exchange protocol combines elliptic curves cryptography with quantum-resistant lattice based public-key cryptographic algorithms. The protocol is designed such that an attack to the protocol able to recover the derived key must attack both the classical and the post-quantum primitive, so the security of the protocol is dependent of the major security of both primitives. In other words, if the attacker has a quantum computer we are protected by the post-quantum algorithms. Conversely, if the chosen post-quantum algorithms turn out not to be secure, we still have classical (and well established) security of the elliptic curves. The protocol presented below is based on the draft of Internet Engineering Task Force about hybrid post-quantum key encapsulation methods (PQ KEM) for transport layer security 1.2 (TLS) [Campagna and Crockett 2021].

For the classical part of the protocol, the library implements an ephemeral elliptic curve Diffie-Hellman using the curve described in Section 4 with the X448 implementation. In this protocol, we obtain the receiver elliptic curve public key PK_r^{ec} and generate

a ephemeral key pair $(SK^{ec}_{eph}, PK^{ec}_{eph})$ using a cryptographic secure random number generator. Then, we compute the elliptic curve shared secret $ss^{ec} = X448(SK^{ec}_{eph}, PK^{ec}_{r})$.

In the post-quantum part, we use the Key Encryption Mechanism provided in Kyber1024, presented in Section 4. Given the receiver post-quantum public key PK^{pq}_r we compute the ciphertext C^{pq} and the post-quantum shared secret ss^{pq} with $C^{pq}, ss^{pq} = ENC(PK^{pq}_r)$ where ENC is the Kyber1024.CCAKEM Key Encapsulation function.

With both shared-secrets, we compute the shared 256-bit symmetric key K_E using the Blake2 function, presented in Section 4, as the keyed hash function $H_K(.)$ such that $K_E = H_{ss^{pq}}(H_{ss^{ec}}(PK^{ec}_{eph}||C^{pq}||PK^{ec}_{r}||PK^{pq}_{r}))$. The protocol returns the symmetric encryption key K_E, the ephemeral elliptic curve public key PK^{ec}_{eph} and the post-quantum encapsulation ciphertext C^{pq}. After encryption of a plaintext with the key K_E, it is necessary to send to the receiver the PK^{ec}_{eph} and C^{pq}, so it can recover K_E for decryption. The protocol is described in Algorithm 1.

The receiver recovers the symmetric key K_E in three steps. First, one executes the elliptic curve Diffie-Hellman using his secret key SK^{ec}_r and the received ephemeral public key PK^{ec}_{eph} obtaining the first shared secret $ss^{ec} = X448(SK^{ec}_r, PK^{ec}_{eph})$. Using the ciphertext C^{pq} and his private key SK^{pq}_r, one is able to recover the second shared secret $ss^{pq} = DEC(C^{pq}, SK^{pq}_r)$ using the Kyber1024.CCAKEM Key Decapsulation function. With both shared secrets it can be obtained $K_E = H_{ss^{pq}}(H_{ss^{ec}}(PK^{ec}_{eph}||C^{pq}||PK^{ec}_{r}||PK^{pq}_{r}))$.

Algorithm 1 Hybrid Key Exchange

function KEY EXCHANGE(PK^{ec}_r, PK^{pq}_r) ▷ Inputs both public keys of the receiver
  $ss^{ec}, PK^{ec}_{eph} \leftarrow ECDHE(PK^{ec}_r)$ ▷ Ephemeral Diffie-Hellman
  $ss^{pq}, C^{pq} \leftarrow ENC(PK^{pq}_r)$ ▷ Post-quantum key encapsulation
  $K_E \leftarrow H_{ss^{pq}}(H_{ss^{ec}}(PK^{ec}_{eph}||C^{pq}||PK^{ec}_{r}||PK^{pq}_{r}))$ ▷ Derived symmetric key
  sends PK^{ec}_{eph}, C^{pq} to receiver
  return K_E ▷ Symmetric Key for encryption
end function

6.3. Symmetric-Key Derivation

This protocol is a Key Derivation Function (KDF) that uses a symmetric-key K_S obtained by the Hybrid Key Exchange protocol to generate one or more secure symmetric keys with no need to call the more expensive asymmetric key protocol. This protocol is based on the *Extract-then-Expand KDF* construction, the same used in the design of HKDF and that has a sound security proof [Krawczyk 2010]. The proposed construction has two differences when compared to HKDF: (i) it has a fixed sized output length, not requiring an extra parameter and a loop in its implementation; and (ii) it uses the hash function Blake2 directly instead of HMAC. Note that both choices improve performance. In particular, HMAC requires a hash function being called two times, therefore, this KDF is at least two times faster. Here, performance was the main goal, as the protocol is designed for a specific use case that consists of the encryption of the biometric data of voters. More specifically, a new key is derived for each voter, yielding in a very computationally intensive process (we have more than 150 million voters in Brazil).

For each voter v, we derive a new symmetric key K_v, using a randomly generated 16-byte salt S_v and the voter's unique registration number R_v. We used the keyed hashing version $H_K(.)$ of Blake2, our choice of hash function, as presented in Section 4. The details are shown in Algorithm 2.

Algorithm 2 Symmetric Key Derivation

function KEY DERIVATION(K_S, R_v, S_v)
  $prk \leftarrow H_{S_v}(K_S)$ ▷ The hash function is used as a randomness extractor
  $K_v \leftarrow H_{prk}(R_v)$ ▷ The hash function is used as a PRF
  return K_v
end function

6.4. Authenticated Encryption

The library's authenticated encryption scheme uses the stream cipher Chacha20 and authentication algorithm Poly1305 described in Section 4. We based our proposed protocol on the ideas contained in the RFC-8439 [Nir and Langley 2018] by the Internet Research Task Force, adapting it for our algorithms.

Our authenticated encryption algorithm receives as input a N-byte long plaintext P_N and a 256-bit symmetric key K_E. As a first step, a 12-byte $Nonce$ is randomly generated using a cryptographic secure random number generator. This random nonce generation aims to prevents nonce misuse, a common problem for implementations using stream ciphers. The plaintext P_N is encrypted with the encryption key K_E and the generated nonce resulting in the ciphertext C_N.

For authentication purposes, we derive a new 256-bit authentication key K_A using the library's keyed hash function with random generated nonce as input and the key K_E. The authenticated array D is the result of the concatenation of the library's version, the nonce used in encryption, the ciphertext C_N and the ciphertext length N as a 8-byte little-endian unsigned integer. We compute the 128-bit authentication tag T applying the MAC algorithm Poly1305 with key K_A at the array D and returns $D||T$. A pseudocode for the authenticated encryption protocol is presented at Algorithm 3.

The random generation of the nonce implies a probability of 2^{-48} for collision. Nevertheless, in the library's use case, a new symmetric key is obtained at each encryption - from the key-exchange protocol (6.2) or from the key derivation scheme (6.3). In this scenario, the key/nonce tuple will have a much smaller probability of being reused.

In the authenticated decryption scheme, the function receives the array D containing the ciphertext and the associated information (library version, nonce and ciphertext length) as presented above, the authentication tag T and the symmetric encryption key K_E. The first step is to recover the authentication key K_A using the nonce and K_E. We check if T is a valid tag for D using K_A, and if not, the algorithm stops and returns an error. Otherwise, we continue the procedure checking if the length N included in D corresponds to the received ciphertext length. Only if both checks pass, we proceed to the decryption of C_N using K_E with our stream cipher and returns the result P_N.

This protocol aims to guarantee that invalid key/tag/ciphertext sets will not be decrypted, reducing the computational cost of invalid function calls.

Algorithm 3 Authenticated Encryption

function AUTHENTICATED_ENCRYPTION(P_N, K_E)
$Nonce \leftarrow CSRNG(12)$ ▷ 12-byte nonce randomly generated
$C_N \leftarrow ENC_{K_E}(P_N, Nonce)$ ▷ Encrypts with stream cipher
$K_A \leftarrow H_{K_E}(Nonce)$ ▷ Auth Key from Nonce and Encryption Key
$N_{le8} \leftarrow num\_to\_8\_le\_byte(length(P_N))$ ▷ Length of plaintext
$V \leftarrow Version()$ ▷ Version of the Library
$D \leftarrow V||Nonce||C_N||N_{le8}$ ▷ Concatenate Associated Data and ciphertext
$T \leftarrow Auth_{K_A}(D)$ ▷ Compute authentication tag
return $D||T$
end function

7. Conclusion

In this work, we presented `libharpia`, a new library to be applied in Brazilian elections. Throughout the work, we provided all design choices and cryptographic primitives, showing that `libharpia` is aligned with the best practices of secure implementations and cryptography. In addition, we detailed the main cryptographic protocols, justifying their designs based on well established literature and standards. We also provided a detailed description of how the library is used in practice, and its API. Through the use of post-quantum cryptography and hybrid protocols, we showed that `libharpia` provides a security advantage when compared with other cryptographic libraries available. Despite all technical characteristics of `libharpia`, the most important aspect of this work is that it consists of another step towards transparency and auditability of the Brazilian elections, describing in a clear and open fashion all details of the library. As a final note, we remember that `libharpia` is present in TSE's TPS, therefore, the source code of the library is available for any group that desires to analyze it and test its security in practice. Additionally, we remark that there is an intention of open-sourcing this library in the near future.

Acknowledgements We would like to thank all TSE's professionals that were involved in the specification and testing of the library.

References

Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Kelsey, J., Liu, Y.-K., Miller, C., Moody, D., Peralta, R., et al. (2020). Status report on the second round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*.

Alashwali, E. S. and Rasmussen, K. (2018). What's in a downgrade? a taxonomy of downgrade attacks in the tls protocol and application protocols using tls. In *International Conference on Security and Privacy in Communication Systems*, pages 468–487. Springer.

Alessandre, S. (2021). Add support for x509 certs with nist p384/256/192 keys. `https://patchwork.kernel.org/project/linux-crypto/cover/20210316210740.1592994-1-stefanb@linux.ibm.com/`.

Aranha, D. F., Barbosa, P., Cardoso, T. N. C., Araújo, C. L., and Matias, P. (2019). The return of software vulnerabilities in the brazilian voting machine. *Comput. Secur.*, 86:335–349.

Bernstein, D. J. (2005). Cache-timing attacks on aes. `https://cr.yp.to/antiforgery/cachetiming-20050414.pdf`.

Bernstein, D. J., Lange, T., and Schwabe, P. (2012). The security impact of a new cryptographic library. In *Progress in Cryptology – LATINCRYPT 2012*, Lecture Notes in Computer Science, pages 159—-176. Springer.

Beullens, W. (2022). Breaking rainbow takes a weekend on a laptop. *IACR Cryptol. ePrint Arch.*, page 214.

Bos, J. W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2018). CRYSTALS - kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P*, pages 353–367. IEEE.

Brunazo Filho, A., Carvalho, M., Teixeira, M., Simplicio Jr, M., and Fernandes, C. (2015). Auditoria especial no sistema eleitoral 2014. *Simpósio Brasileiro em Segurança da Informação e de Sistemas Computactionais, XV, Florianópolis. Anais... Florianópolis: SBC*, pages 511–522.

Campagna, M. and Crockett, E. (2021). Hybrid post-quantum key encapsulation methods (pq kem) for transport layer security 1.2 (tls). Internet-draft, IETF Secretariat. `https://www.ietf.org/archive/id/draft-campagna-tls-bike-sike-hybrid-07.txt`.

Coimbra, R. C. M., Monteiro, J. R. M., and da Silva Costa, G. (2017). Registro impresso do voto, autenticado e com garantia de anonimato. *Anais do XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 666–681.

Coutinho, M. and Neto, T. C. S. (2021). Improved linear approximations to ARX ciphers and attacks against chacha. In *Advances in Cryptology - EUROCRYPT 2021*, volume 12696 of *Lecture Notes in Computer Science*, pages 711–740. Springer.

Denis, F. (2013). The sodium cryptography library. `https://download.libsodium.org/doc/`.

Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., and Stehlé, D. (2018). Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268.

Gallo, R., Kawakami, H., Dahab, R., Azevedo, R., Lima, S., and Araujo, G. (2010). T-DRE: a hardware trusted computing base for direct recording electronic vote machines. In *Twenty-Sixth Annual Computer Security Applications Conference, ACSAC 2010, Austin, Texas, USA, 6-10 December 2010*, pages 191–198. ACM.

Hamburg, M. (2015). Ed448-goldilocks, a new elliptic curve. *IACR Cryptol. ePrint Arch.*, page 625.

Krawczyk, H. (2010). Cryptographic extraction and key derivation: The HKDF scheme. In *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648. Springer.

Langley, A. (2010). ctgrind—checking that functions are constant time with valgrind. https://github.com/agl/ctgrind.

Leurent, G. and Pernot, C. (2021). New representations of the AES key schedule. In *Advances in Cryptology - EUROCRYPT 2021*, volume 12696 of *LNCS*, pages 54–84. Springer.

Lyubashevsky, V. (2009). Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *Advances in Cryptology - ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616. Springer.

MATZOV (2022). Report on the Security of LWE: Improved Dual Lattice Attack.

Monteiro, J., Lima, S., Rodrigues, R., Alvarez, P., Meneses, M., Mendonça, F., and Coimbra, R. (2019). Protegendo o sistema operacional e chaves criptográficas numa urna eletrônica do tipo t-dre. In *Anais do IV Workshop de Tecnologia Eleitoral*, pages 1–12. SBC.

Nir, Y. and Langley, A. (2018). ChaCha20 and Poly1305 for IETF Protocols. RFC 8439.

NIST (2022). Nist announces first four quantum-resistant cryptographic algorithms. https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms.

Novark, G. and Berger, E. D. (2010). Dieharder: securing the heap. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010*, pages 573–584. ACM.

Pointcheval, D. (2006). *Topics in Cryptology – CT-RSA 2006: The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2005, Proceedings*. LNCS sublibrary: Security and cryptology. Springer.

Regev, O. (2005). On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM.

Roetteler, M., Naehrig, M., Svore, K. M., and Lauter, K. (2017). Quantum resource estimates for computing elliptic curve discrete logarithms. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 241–270. Springer.

Saarinen, M. O. and Aumasson, J. (2015). The BLAKE2 cryptographic hash and message authentication code (MAC). *RFC*, 7693:1–30.

Tromer, E., Osvik, D. A., and Shamir, A. (2010). Efficient cache attacks on aes, and countermeasures. In *Journal of Cryptology*, volume 23, pages 37–71.

TSE (2021). Novas urnas eletrônicas contarão com certificação da icp-brasil. https://www.tse.jus.br/comunicacao/noticias/2021/Julho/novas-urnas-eletronicas-contarao-com-certificacao-da-icp-brasil.

van de Graaf, J. and Custódio, R. (2002). Tecnologia eleitoral e a urna eletrônica–relatório sbc 2002. *Disponıvel em http://www.sbc. org.br/index.php*.