Received: 14 May 2019 | Revised: 1 April 2020 | Accepted: 10 April 2020
DOI: 10.1002/cpe.5813

WILEY

SPECIAL ISSUE PAPER

Design and practical implementation of verify-your-vote protocol

Marwa Chaieb[1] | Souheib Yousfi[2] | Pascal Lafourcade[3] | Riadh Robbana[2]

[1]LIPSIC, Faculty of Sciences of Tunis, University Tunis El-Manar, Tunis, Tunisia

[2]LIPSIC, National Institute of Applied Science and Technology, University of Carthage, Tunis, Tunisia

[3]LIMOS, University Clermont Auvergne, Clermont-Ferrand, France

Correspondence
Marwa Chaieb, LIPSIC, Faculty of Sciences of Tunis, University Tunis El-Manar, Tunis, Tunisia.
Email: chaiebmarwa.insat@gmail.com

Summary

One of the most critical properties that must be ensured to have a secure electronic voting is verifiability. Political parties, observers, and especially voters want to be able to verify that all eligible votes are cast as intended and counted as cast without compromising votes secrecy or voters privacy. Over the past few decades, an important number of e-voting protocols attempt to deal with this issue by using cryptographic techniques and/or a public bulletin board. Recently, some blockchain-based e-voting systems have been proposed, but were not found practical in the real world, because they do not support situations with large numbers of candidates and voters. In this article, we design and implement a verifiable blockchain-based online voting protocol, called verify-your-vote . Our protocol ensures several security properties thanks to some cryptographic primitives and blockchain technology. We also evaluate its performance in terms of time, cost, and the number of voters and candidates that can be supported.

KEYWORDS

blockchain technology, cryptographic primitives, e-voting, practical implementation, security and performance evaluation

1 | INTRODUCTION

Voting is a crucial aspect of democracy. Secure and transparent elections have always been the major concern of voters and political parties. In traditional elections, every voter needs to physically go to a polling station to cast a vote and all votes are counted manually by the election organizers, who may detect and commit errors in the final election tally. To reduce these inherent voting limitations and due to recent cryptographic improvements, these technologies have been developed. Called electronic voting simply e-voting, these systems have several advantages, including the simplicity of the process, automated tallying, and the reduction of organizational costs. The security provided by such a system should include, at least, the following properties[1]: *Eligibility:* only registered voters can vote and only one vote per voter is counted (if the voter is allowed to vote more than once, the most recent ballot will be tallied and all others are discarded); *Individual verifiability:* the voter him/herself must be able to verify that his/her ballot was counted correctly; *Universal verifiability:* the election official tally must be verifiable by all parties; *Vote-privacy:*the connection between a voter and his/her vote cannot be established without his/her help; *Receipt-freeness:* a voter cannot prove to a potential coercer that he/she voted in a particular way; *Coercion resistance:* even when a voter interacts with a coercer during the voting process, the coercer will not be sure of whether the voter obeyed his demand or not; *Integrity:* ballots are not altered or deleted during any step of the election; *Fairness:* no partial results that could influence voters are published before the official tally; *Robustness:* the system should be able to tolerate some faults, *Vote-and-go:* a voter does not need to wait for the end of the voting phase or trigger the tallying phase; *Voting policy:* specify if a voter can vote only once or has the possibility to change his/her choice before the end of the election. On the other hand, existing e-voting systems suffer from several security issues since they are centralized by design. Hence, to achieve the trustworthiness required by voters and election organizers, e-voting systems must be secure, while allowing transparency of elections. Blockchain, which is a distributed public ledger, helps to achieve this level of security and verifiability, while

maintaining confidentiality and nonmalleability of transactions. Indeed, it operates without the need for a trusted central authority and ensures data integrity as every transaction is verified and stored by each of the nodes in the blockchain. All transactions are gathered into blocks by miners, who must complete a proof such as a proof-of-work or proof-of-stake. Thus, blockchain is considered an immutable and secure data structure. In this article, we design and implement a secure and verifiable blockchain-based e-voting protocol called verify-your-vote.[2]

- Contributions: Our contributions can be summarized as follow:
  - Design and practical implementation of a secure and verifiable blockchain based e-voting system, called verify-your-vote (VYV),
  - Evaluation of the performance of the protocol in terms of cost, time, and the number of voters and candidates that can be supported.
  - Formal and informal security evaluation of VYV protocol.
- Article organization: We study some existing e-voting systems in Section 2 and discuss the cryptographic techniques used in VYV in Section 3. In the next section, we present the different phases of VYV protocol and give the details of its practical implementation. We then analyze its performance and evaluate its security in Sections 5 and 6, respectively. Section 7 is a conclusion and a set of perspectives.

2 | RELATED WORK

In the last few years, a number of online blockchain-based e-voting systems have been proposed to resolve the security problems of traditional voting protocols. In this section, we give an overview and evaluate some of these systems.

A smart contract for boardroom voting with maximum voter privacy[3](OVN): It is a self-tallying voting protocol based on blockchain technology. It is implemented on the Ethereum blockchain and supports only elections with two options "yes" or "no". OVN allows voters and any third party to compute the election final result without assistance which guarantees voter's privacy. In return, it does not ensure fairness. In fact, it is possible for voters that have not yet cast their votes to cooperate and compute the election partial results. If these voters are dissatisfied with the tally, they can simply either not cast votes or send invalid votes and in both cases the other voters cannot perform the final tally. Thus, this protocol does not ensure robustness. Also, it is not coercion resistant and supports only elections with a maximum of 50 voters due to the mathematical tools they have used. Finally, it needs to trust the election administrator to register and authenticate eligible voters.

Platform-independent secure blockchain-based voting system[4](PSBVS): It is an independent e-voting system implemented on a BFT consensus based blockchain. Authors claim that their solution does not rely on a centralized trusted party to compute and publish the election final result, but they still need to trust an administrator to decrypt the sum of votes and upload the result to the blockchain. They use Paillier cryptosystem[5] to encrypt votes before publishing them, proof of knowledge to ensure the correctness and consistence of votes, and short-linkable ring signature (SLRS) to guarantee voters privacy. This protocol does not ensure voters eligibility since a voter can register him/herself by simply providing his/her e-mail address, ID number, or an invitation URL with a password and these mechanisms are not sufficient to verify the eligibility of a voter. In addition, this protocol does not respect the definition of coercion resistance given by Juels et al.[6] A coercer can vote in the place of a voter if he knows the voter's secret key. The coerced voter cannot provide a fake secret key to the coercer because a vote with a fake secret key is rejected by the smart contract. In summary, the security of this protocol relies on the honesty of the administrator, the smart contract, and the blockchain validation nodes that replicate the execution of the smart contract codes to ensure its correct execution. To achieve the trustworthiness of the blockchain platform, the authors propose to allow different parties to host the blockchain validation nodes. However, the major concern of the BFT consensus based blockchain is the scalability of nodes. As shown in articles,[7,8] hyperledger stops working beyond 16 nodes. Due to this limitation, the authors implement their protocol on the hyperledger fabric blockchain using only four validation nodes for an election with one million voters.

An end-to-end voting-system based on bitcoin[9] (*EtEVBB*): Based on blockchain technology, this voting platform uses bitcoin as a ballot box to ensure an end-to-end verifiability. This solution uses an anonymous Kerberos authentication protocol[10,11] to authenticate voters and ensure their anonymity. It also uses the digital asset coin based on the Open Asset Protocol (OAP) to create voting tokens. To vote, each eligible voter transfer his/her voting token to the bitcoin address of the chosen candidate. The final count is then obtained by summing the tokens received by each candidate. Authors mentioned that the count should start after 80 minutes of the election closing time due to the time that take the mining process of Bitcoin (10 minutes per block) and to avoid forks. This protocol does not provide any mechanism to avoid coercion resistance or to ensure receipt-freeness. Finally, all parties can watch the election progress in real time and get partial results.

End-to-end voting with non-permissioned and permissioned ledgers[12] (*EtEVnPPL*): Authors extend the article[9] by proposing another implementation of the same e-voting protocol using a different blockchain platform, which is the multichain. They started by recalling the different phases of the protocol and its implementation over bitcoin. Then, they described possible threats related to bitcoin that impact on the security of their proposed protocol, such as DDoS attacks, software bugs on wallets, misbehavior of pool of miners, and some other problems related to the anonymity in bitcoin. For those reasons, they proposed to use multichain, a platform for the creation and deployment of private blockchains, derived from bitcoin core. This platform restricts the access to the blockchain to only chosen participants, introduces controls over which transactions are permitted and avoid the proof of work consensus algorithm. Authors created two blockchains: the first one is dedicated to manage the voting process and the second one to manage the anonymous ID of voters instead of using anonymous Kerberos authentication or blind signature. Authors claim that their

new proposed solution guarantees all the security requirements proposed by Bistarelli et al.[9] and adds two other security properties: (i) uncoercibility and receipt freeness and (ii) data confidentiality and neutrality. However, referring to the definition of coercion resistance introduced by Juels et al.,[6] this new implementation is not coercion resistant because a coercer can vote in the place of the coerced voter if this latter provides his/her private key to the coercer or if the coercer is next to the voter and controlling him/her during the voting phase.

Follow my vote[1] (*FMV*): It is a commercial voting system that employs the blockchain as a ballot box. To authenticate themselves, voters need a web cam and an ID. They can verify voting progress in real time. This possibility compromises the fairness of the election. In addition, FMV requires a trusted authority to hide the correspondence between voters identities and their voting key. This authority also has the possibility to change votes since it has all voters' pass-phrases. Also, votes secrecy is not verified by this system as votes are cast without being encrypted. Furthermore, this system does not offer any mechanism to allow voters or observers to check the accuracy of the election final result.

TIVI[2]: It is a commercial solution proposed by a Venezuelan-owned multinational company specializing in technology solutions for governments, named Smartmatic. This online voting system includes an authentication phase based on biometric techniques. Indeed, voters are authenticated by means of a selfie. TIVI uses cryptographic primitives to ensure votes privacy and voters anonymity. Taking advantage of the blockchain technology, it ensures universal verifiability and votes integrity. However, this voting system is not coercion resistant and does not ensure receipt-freeness.

3 | PRELIMINARIES

In this section, we introduce the technologies and the cryptographic primitives used in our protocol.

Blockchain technology: A public and decentralized ledger that operates without a central authority. It stores the different exchanges made between its users in a transparent and secure way. To ensure data integrity, all the nodes on the blockchain verify and store every transaction. Transactions are then gathered into blocks by miners. For every new block, it is required to follow a consensus mechanism, such as Proof-of-Work (PoW) and Proof-of-Stake (PoS), to agree on the next block to be appended to the chain. The blockchain technology has an append-only data structure, such that new blocks can be written to it but cannot be altered or deleted. In order to rewrite a part of the blockchain, the majority of the computational power on the network (at least 51%) would need to collude.

Ethereum blockchain[3]: A decentralized, open source and public computing platform based on blockchain technology. It expands the functionalities of blockchain by implementing smart contracts. It offers a tuning complete virtual machine called Ethereum virtual machine (EVM), where smart contracts can be run. There are two types of account in Ethereum: (i) externally owned account (EOA), which is a user-controlled account, characterized by a key pair that allows the user to send and receive transactions and (ii) smart contract account, which is a set of code stored on the blockchain. To protect the system against malicious users and compensate miners for their computational power usage, the execution of each transaction includes a transaction fee, called "gas". Gas is the unit of measure for the amount of work that is accomplished for an operation and gas price is measured in terms of Ether, which is the cryptocurrency of Ethereum.

Elliptic curve: A geometric curve that has particularly interesting properties for the world of cryptography. To add two points P and Q of an elliptic curve, it is enough to remark that in certain cases, the line L passing through these two points also passes through a third point R' of the curve. The result of the addition will be represented by the symmetrical point of R.'

Elliptic curve cryptography (ECC)[13]: In cryptography, elliptic curves are used for asymmetric operations. One of the main advantage of ECC is its efficiency compared with traditional cryptography because it offers equal security level for a far smaller key size.

Pairings[14]: Another advantage of elliptic curve cryptography is that a bilinear operator can be defined between groups. Let G_1 be an additive cyclic group of order a prime number q and G_2 a multiplicative group of the same order q. A function $e{:}G_1{\times}G_1{\rightarrow}G_2$ is called a bilinear cryptographic coupling (also called pairing) and denoted by $e(.,.)$, if it satisfies the following properties:

1. Bilinearity: for all $P, Q \in G_1$ and $a,b{\in}Z; e(aP,bQ){=}e(P,Q)^{ab}$.
2. Nondegeneration: $e(P,P)$ is a generator of G_2 and so $e(P,P){\neq}1$.
3. Computability: there is an efficient algorithm to compute $e(P,Q)$ for all $P,Q{\in}G_1$.

Identity-based encryption (IBE): It is an asymmetric cryptosystem where the public key of a user is an arbitrary string related to its identity for example: an e-mail address, a phone number, or an IP address, and the secret key is given by a trusted authority. ID-based encryption was initially proposed by Adi Shamir[15] in 1984 and the first IBE protocol was proposed in 2001 by Boneh and Franklin.[16] In this protocol, the private key generator (PKG) knows the private key. This issue can be corrected by using a distributed PKG like that of Pedersen[4] or Gennaro et al.[17] In these protocols, a master key is generated in a way whereby each of the *m* PKGs randomly constructs a fragment. Such a scheme unfolds in four steps that we represent by the following functions:

[1] https://followmyvote.com/
[2] https://www.smartmatic.com/elections/online-voting/tivi/
[3] https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf
[4] https://www.cryptoworkshop.com/ximix/lib/exe/fetch.php?media=pedersen.pdf

Ballot number BN			
Pseudo ID "C_j"	Candidate's name "$name_j$"	Choice	Counter-value "$CV_{BN,name_j,k}$"
0	Paul	☐	$CV_{BN,name_0,0}$
1	Nico	☐	$CV_{BN,name_1,1}$
2	Joel	☐	$CV_{BN,name_2,2}$

FIGURE 1 Ballot structure

1. *Setup():* the PKG generates the different parameters namely two groups $\mathbf{G}_1$ and $\mathbf{G}_2$ of order a prime number q, a generator $P \in \mathbf{G}_1$, a pairing function $e(.,.)$, two hash functions H_1 and H_2 as well as the master secret key msk and its corresponding master public key $mpk = msk \cdot P$.
2. *Extract(msk, id):* the PKG calculates the secret key related to the identity $\varepsilon id \varepsilon$ using the following formula $sk = msk \cdot H_1(id)$.
3. *Encrypt(mpk, id, Msg):* We use the mpk of the PKG and the id of the receiver to calculate the cipher-text of a given message "Msg". We use the following formula: $EncMsg = (r \cdot P, Msg \oplus H_2(g_{id}^r))$, where $r \in \mathbf{Z}_q^*$ and $g_{id} = e(H_1(id), mpk)$.
4. *Decrypt(sk, EncMsg):* The receiver uses his secret key do decrypt $EncMsg = (u,v)$ by using the following formula: $Msg = v \oplus H_2(e(msk,u))$.

Paillier cryptosystem[18]: Proposed by Pascal Paillier in 1999, it is a nondeterministic asymmetric algorithm for public key cryptography. It is based on computations over the group $\mathbb{Z}_{n^2}^*$, where n is an RSA modulus. This scheme has the additive homomorphism property, which allows the encryption of many bits in one operation with a constant expansion factor, and allows for efficient decryption.

4 | VERIFY-YOUR-VOTE PROTOCOL: DESIGN AND IMPLEMENTATION

In this section, we give a detailed description of verify-your-vote[2] protocol, an online electronic voting protocol that uses Ethereum blockchain as a public bulletin board and is based on a variety of cryptographic primitives, namely ECC,[13] pairings,[14] and IBE.[16] We start by describing the structure of a ballot and the signification of each parameter in the ballot, we present then the list of protocol entities as well as their roles during the voting process and finally we give the different phases of the protocol with their implementation details.

4.1 | Ballot structure

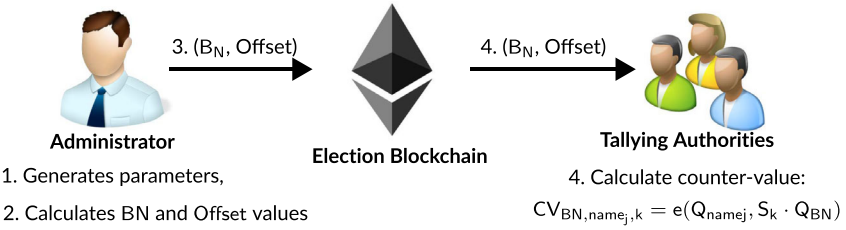
As illustrated in Figure 1, each ballot is composed of a unique bulletin number BN obtained as follows: $BN = E_{PK_A}(g, D)$, where g is a generator of an additive cyclic group G, D is a random number, and $E_{PK_A}(x)$ denotes the encryption of the message x with the administrator public key PK_A, using Paillier cryptosystem. Each ballot also contains a set of m candidates name $name_j$ and candidates pseudo ID, denoted C_j, which are the positions of candidates on the ballot, obtained from an initial order and an offset value. The offset value is obtained, for each ballot, by using the following formula: $Offset = H(g) \bmod m$, where H is an hash function, g is the same generator used to calculate the ballot number BN, and m is the number of the candidates who participate to the election. In addition, each ballot includes a set of countervalues $CV_{BN,name_j,k}$ that serves as a receipt for each voter. They are obtained by using the following formula: $CV_{BN,name_j,k} = e(Q_{namej}, S_k \cdot Q_{BN})$, where S_k is the secret key of the tallying authority TA_k, $Q_{namej} = H_1(name_j)$ and $Q_{BN} = H_1(BN)$ are two points of the elliptic curve E, and H_1 is an hash function.

4.2 | Protocol entities

Five entities interact with each other during our voting process. They are:

- *A registration authority (RA)* that verifies the eligibility of voters by a face-to-face meeting and gives access to only eligible voters to a registration server,
- *A registration server (RS)* that provides only eligible electors with their authentication parameters,
- *An administrator (Admin)* that manages the election, sets up its different parameters, authenticates voters, and participates in the construction of ballots,
- *A set of n eligible voters (V_i)* who have the right to vote more than once and only their last votes will be counted. They have the possibility to verify that their votes were counted correctly and check the accuracy of the election final result,
- *A set of m tallying authorities (TAs)* that construct ballots, decrypt votes, perform the tally, and publish the results on the blockchain.

FIGURE 2 Setup phase



4.3 | Voting process and implementation

VYV protocol unfolds in six steps. In this part, we describe each phase and give its implementation details. We start by presenting the software and hardware environments of the protocol implementation.

1. *Hardware environment:* We deploy the protocol on a personal computer with an Intel(R) Core(TM) i5-3210M processor with 4 GB RAM and running on Ubuntu 16.04.
2. *Software environment:* For the implementation of our system, we choose an interpreted, object-oriented, high-level, and general-purpose programming language, which is *Python*.[5] We also use a set of cryptographic libraries, written in Python, to develop the cryptographic primitives of our protocol. These libraries are: *Charm Crypto*[19]:a framework for rapidly prototyping advanced cryptosystems, ships with a library of implemented cryptosystems. We import from this framework the package of identity-based encryption to be used to encrypt and decrypt votes; *Cryptography*[6]: provides a variety of cryptographic primitives. From this package, we use the elliptic curve cryptography (ECC) scheme. We choose the *SECP256K1* elliptic curve, defined in Standards for Efficient Cryptography (SEC)[7]; *Tate_bilinear_pairing 0.6*[8]:allows to compute the Tate bilinear pairing. We use this package to calculate countervalues. We recall here the formula of a countervalue: $CV_{BN,name_j,k} = e(Q_{name_j}, S_k \cdot Q_{BN})$; *PHE*[9]: A Python library for partially homomorphic encryption. We use it to generate the voters and the administrator key pairs, to encrypt and to decrypt the authentication parameters of eligible voters; and *Hashlib:* implements many different secure hash and message digest algorithms. From this package, we use the SHA256 hash algorithm to calculate the Offset value of each ballot (Offset=H(g) mod m).

We present now the different phases of our approach as well as the set of Python functions developed to implement each phase.

1. *Online setup phase*: This phase is illustrated by Figure 2. The administrator generates the election parameters and publishes them on the blockchain. These parameters are G_1 an additive cyclic group of order a prime number q, G_2 a multiplicative group of the same order q, an Hash function: H:{0,1}*→G_1, and PK_A the administrator public key, generated using Paillier's cryptosystem. Then, the administrator calculates each ballot number and its corresponding offset value ($BN = \{g, D\}_{PK_A}$, Offset= H(g) mod m) and sends them to the TAs via the blockchain to obtain for a given tallying authority TA_k the corresponding countervalue ($CV_{BN,name_j,k} = e(Q_{name_j}, S_k \cdot Q_{BN})$).

To implement this phase, we develop the function described by Algorithm 1.

Algorithm 1. *ConstructBallots*

INPUT: *l*, *m*, *Names*[ ], S_k, G_1, *n*
OUTPUT: *ListBallots*[ ]
For i ← 1 to *l*
  D ← random number;
  g ← generator from (G_1, *n*);
  BN ← PaillierEncrypt(g||D, PK_A);
  Offset ← H(g) mod m;
  For j ← 1 to *m*
    $CV_{BN,name_j,k}$ ← pairing($H(Names[j])$, $S_k \cdot H(BN)$);
    ListCV ← append($CV_{BN,name_j,k}$)
  end For
  Ballot = [*BN*, *Offset*, *Names*[ ], *ListCV*[ ]]
  ListBallots[ ] ← append(*Ballot*)
end For
return(*ListBallots*[ ]);

[5]https://www.python.org/
[6]https://pypi.org/project/cryptography/
[7]http://www.secg.org/sec2-v2.pdf
[8]https://pypi.org/project/tate_bilinear_pairing/
[9]https://pypi.org/project/phe/

- ConstructBallots (Algo1): This function takes in input the voters number l, the candidates number m, a list of their names $Names[\ ]$, the secret key of the tallying authority S_k, the group G_1 and its order n and returns a list of l ballots.

2. *Off-line registration phase*: The voters physically go to a polling station to register and obtain their authentication parameters. To verify his/her eligibility, each voter must provide his/her ID card to the registration authority who checks if the voter is eligible to participate in the election. Then, only eligible voters have access to a registration server to get their authentication parameters. Each voter's credentials have the following form: $(S_{PW_i} = S_{RS\_A}.H_1(PW_i), P_{PW_i} = H_1(PW_i))$, where PW_i is a password entered by the voter V_i and S_{RS_A} is a secret value shared between the RS and the Admin. This phase is illustrated by Figure 3.
3. *Online authentication phase*: All registered voters sign and encrypt their authentication parameters by using the administrator public key PK_A and send them to the admin who decrypts and checks the validity of each voter authentication parameters. If the voter is eligible to vote, he/she has the right to create his/her own account on the election blockchain. From this account, he/she participates in the election process and publishes his/her public key. Figure 4 illustrates these interactions. To implement this phase we develop two functions described by Algorithms 2 and 3.
   - EncryptAuthParam (Algo2): This function encrypts the login and the password of a voter. It takes as input the administrator public key PK_A, the login *Login* and the password *PWD* of the voter and outputs the encrypted login and password (*EncLogin*, *EncPWD*).
   - VerifyAuthParam (Algo3): This function implements the role of the administrator during the authentication phase. It takes as input an encrypted login *EncLogin*, an encrypted password *EncPWD*, the administrator secret key SK_A and the shared secret S_{RS_A}. It returns *True* if the authentication parameters are valid and *False* otherwise.

Algorithm 2. *EncryptAuthParam*

INPUT: PK_A, *Login*, *PWD*
OUTPUT: *EncLogin*, *EncPWD*
1: *EncLogin* ← *PaillierEncrypt*(*Login*, PK_A);
2: *EncPWD* ← *PaillierEncrypt*(*PWD*, PK_A);
3: return(*EncLogin*, *EncPWD*);

Algorithm 3. *VerifyAuthParam*

INPUT: *EncLogin*, *EncPWD*, SK_A, S_{RS_A}
OUTPUT: A boolean value
1: DecLogin ← PaillierDecrypt(EncLogin, SK_A);
2: DecPWD ← PaillierDecrypt(EncPWD, SK_A);
3: If DecLogin = S_{RS_A}· DecPWD
4: return(True);
5: else
6: return(False);
7: end if

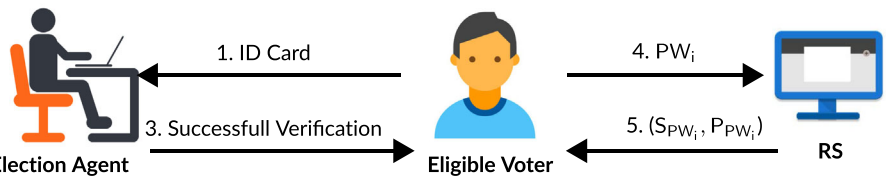


FIGURE 3 Registration phase

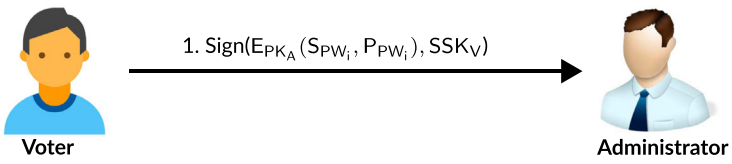


FIGURE 4 Authentication phase

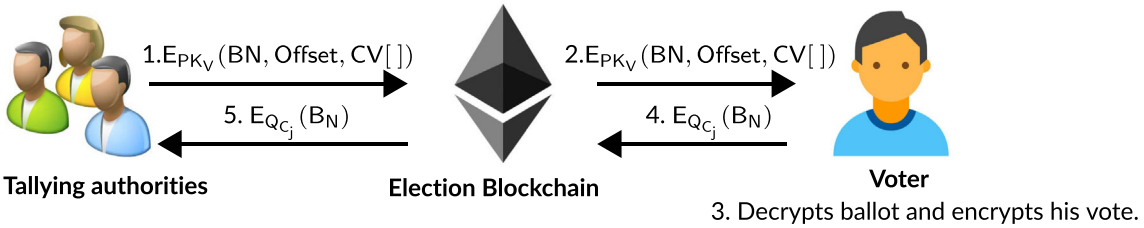


FIGURE 5 Voting phase

4. *Online voting phase*: As presented in Figure 5, two entities participate during this phase:
   - TAs: They randomly choose a ballot for each voter, encrypt it with the voter's public key and send it to the corresponding voter via the blockchain.
   - Eligible voters: When receiving his/her ballot, each voter proceeds to the decryption of his/her ballot, chooses his/her favorite candidate, encrypts his/her vote, casts it to the TAs via the blockchain and saves the corresponding countervalue $CV_{BN,name_j,k}$ which allows him/her to verify his/her vote later. To encrypt his/her vote, each voter chooses a candidate with pseudo ID C_j and encrypts it using his/her ballot number *BN*. Thus, each encrypted vote has the following form: $Enc\_Vote = E_{Q_{C_j}}(BN)$ where $Q_{C_j} = H_1(C_j)$.

   To process this phase, we implement two functions described by Algorithms 4 and 5.

Algorithm 4. *EncryptBallot*

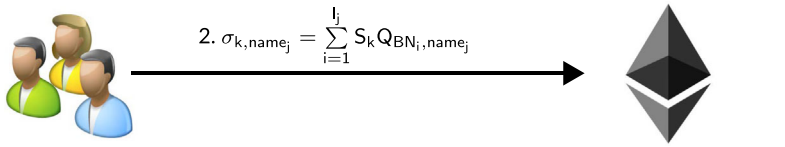
INPUT: $Ballot = [BN, Offset, Names[\,], ListCV[\,], PK_V]$
OUTPUT: $EncBallot = [EncBN, EncOffset, Names[\,], ListCV[\,]]$
1: EncBallot[ ] ← Ballot[ ];
2: EncBallot[0] ← PaillierEncrypt(Ballot[0], PK_V);
3: EncBallot[1] ← PaillierEncrypt(Ballot[1], PK_V);
4: return(EncBallot[ ]);

Algorithm 5. *Vote*

INPUT: $EncBallot[\,], MPK, SK_V$
OUTPUT: $EncVote, CV_{BN,name_j,k}$
1: Ballot[ ] ← EncBallot[ ];
2: Ballot[0] ← PaillierDecrypt(EncBallot[0], SK_V);
3: Ballot[1] ← PaillierDecrypt(EncBallot[1], SK_V);
4: C_j ← input of the index of the chosen candidate;
5: *BN* ← Ballot[0];
6: EncVote ← Encrypt(*MPK*, C_j, *BN*);
7: $CV_{BN,name_j,k}$ ← Ballot[3][C_j];
8: return(EncVote, $CV_{BN,name_j,k}$);

   a. EncryptBallot (Algo4): Implements the role of TAs during the voting phase. It takes in input a ballot and an eligible voter public key and returns the encrypted ballot.
   b. Vote (Algo5): This function allows voters to vote for a certain candidate. It takes as input an encrypted ballot and the master public key of the IBE scheme *MPK*, decrypts the ballot using the voter secret key SK_V and outputs an encrypted vote and its corresponding counter value. To encrypt a vote, we use the identity-based encryption scheme, imported from the Charm package. A ballot is represented by the following list: $Ballot = [BN, Offset, [name_1, name_2, \ldots, name_m], [CV_{BN,name_1,1}, CV_{BN,name_2,2}, \ldots, CV_{BN,name_m,m}]]$.

5. *Online tallying phase*: TAs decrypt all eligible votes using their secret keys to obtain the bulletin numbers *BN*. Each tallying authority is dedicated to calculate the number of votes of a specific pseudo *ID* (C_j): for example the first tallying authority $\varepsilon TA_1 \varepsilon$ decrypts, with its secret key $S_1 \cdot Q_{C1}$, all bulletins that were encrypted with the public key Q_{C1}. Each TA is responsible for generating its own secret key to decrypt votes by executing the function *Extract* of the IBE scheme (see Section 3). This means that each TA plays the role of a PKG. This step can be performed before the beginning of the election. From each ballot number *BN* and its corresponding offset value, TAs reconstruct ballots, identify chosen candidates and add up the counters. Once all votes have been decrypted and counted, TAs publish, on the election blockchain, the final result of the election



1. Decrypts votes to reveal bulletin numbers, Reconstructs ballots, Identifies chosen candidates and increments counters.

FIGURE 6 Tallying phase

as well as the count of each candidate using the following formula: $\sigma_{k,name_j} = \sum_{i=1}^{I_j} S_k \cdot Q_{BN_i,name_j}$, where I_j is the number of votes received by the candidate with name $name_j$, S_k is the private key of the tallying authority k, $Q_{BN_i,name_j} = H_1(BN_{i,name_j})$ and $BN_{i,name_j}$ is the ballot number of the vote i that corresponds to the candidate with name $name_j$. This phase is described by Figure 6 and implemented by two functions given by Algorithms 6 and 7.

- Tally (Algo6): This function takes in input an encrypted vote *EncVote*, the secret key of the TA S_K, the list of ballot numbers *ListBN*[ ], the list of their corresponding offset values *ListOffset*[ ], and the list of candidates name *Names*[ ]. It returns the result of the election after incrementing counters.
- CalculSigma (Algo7): This function implements the formula $\sigma_{k,name_j} = \sum_{i=1}^{I_j} S_k \cdot Q_{BN_i,name_j}$ (lines 2, 3 ,and 4 of Algo7) to calculate the count of each candidate. It takes in input a candidate's name $name_j$, the list of ballot numbers that contains a vote for this candidate *ListBN*[ ] and the tallying authority secret key S_k.

Algorithm 6. *Tally*

INPUT: *EncVote*, S_K, *ListBN*[ ], *ListOffset*[ ], *Names*[ ]
OUTPUT: *Result*
1: DecVote ← Decrypt(S_K, EncVote);
2: For i ← 1 to length(*ListBN*)
3: If (DecVote = *ListBN*[i])
4: Offset ← *ListOffset*[i];
5: end If
6: end For
7: Result ← *IncrementCounter*(*Offset*, *Names*[ ]);
8: return(Result);

Algorithm 7. *CalculSigma*

INPUT: $name_j$, *ListBN*[ ], S_k
OUTPUT: $\sigma_{k,name_j}$
1: $\sigma_{k,namej} \leftarrow 0$;
2: For i ← 1 to length(*ListBN*[ ])
3: $\sigma_{k,name_j} \leftarrow \sigma_{k,name_j} + S_k \cdot H_1(ListBN[i])$;
4: end For
5: return($\sigma_{k,name_j}$);

6. *Online verification phase*: This phase allows voters to verify that their votes have been cast as intended (individual verification) and counted as cast (universal verification). It includes two sub-phases:
   - During the first sub-phase, TAs recalculate countervalues from each ballot number and the chosen candidate's name and publish them on the blockchain. Thus, every voter who wishes to verify that his/her last vote has been included in the final tally, can access the election blockchain and check the existence of his/her countervalue that he/she saved during the voting phase, in the list of reconstructed countervalues. This subphase is illustrated by Figure 7 and implemented by the following functions described by Algorithms 8 and 9:
     - ReconstructCV (Algo8): This function takes in input a ballot number, the name of the chosen candidate, and the secret key of the tallying authority S_k. It returns the corresponding countervalue $CV_{BN_i,name_j,k}$.
     - VerifyCV (Algo9): This function takes as inputs the list of reconstructed countervalues *ReconstructedCV*[ ] as well as the voter's receipt *CV* and checks its existence in the list. It returns *True* if *ReconstructedCV*[ ] contains the value of *CV* and *False* otherwise.

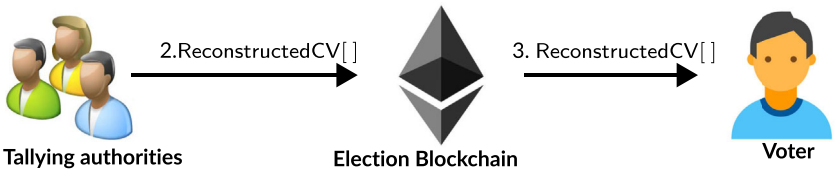


FIGURE 7 Verification first subphase

- The second sub-phase allows voters and all other parties to check the accuracy of the final result from the list of reconstructed countervalues CV_{BN_i} and the count of each candidate as follows:

$$\prod_{i=1}^{l} CV_{BN_i} = \prod_{k=1}^{m}\prod_{j=1}^{m}\prod_{i=1}^{l_j} CV_{BN_{i,name_j},k} = \prod_{k=1}^{m}\prod_{j=1}^{m}\prod_{i=1}^{l_j} e(Q_{name_j}, S_k.Q_{BN_i,name_j}) = \prod_{k=1}^{m}\prod_{j=1}^{m} e(Q_{name_j}, \sum_{i=1}^{l_j} S_k.Q_{BN_i,name_j})$$
$$= \prod_{k=1}^{m}\prod_{j=1}^{m} e(Q_{name_j}, \sigma_{k,name_j}) \quad (1)$$

where $l = \sum_{j=1}^{m} l_j$ is the total number of votes. These equalities use the bilinear property of pairing:

$$\prod_{i=1}^{l_j} e(Q_{name_j}, S_k \cdot Q_{BN_i,name_j}) = e(Q_{name_j}, \sum_{i=1}^{l_j} S_k \cdot Q_{BN_i,name_j})$$

We develop the function described by Algorithm 10 to implement this sub-phase:

- VerifyResult (Algo10 parts 1 and 2): It takes as input the list of reconstructed countervalues *ReconstructedCV*[ ], the list of candidates count $\sigma_{k,name_j}$[ ][ ], and the list of candidates name *Names*[ ], and outputs *True* if the equation 1 is verified and *False* otherwise.

Algorithm 8. *ReconstructCV*

INPUT: BN_i, $name_j$, S_k
OUTPUT: $CV_{BN_i,name_j,k}$
1: $Q_{BN} \leftarrow H_1(BN)$;
2: $Q_{name_j} \leftarrow H_1(name_j)$;
3: $CV_{BN_i,name_j,k} \leftarrow$ pairing$(Q_{name}, S_k \cdot Q_{BN})$;
4: return$(CV_{BN_i,name_j,k})$;

Algorithm 9. *VerifyCV*

INPUT: *ReconstructedCV*[ ], *CV*
OUTPUT: A boolean value
1: For i ← 1 to length(*ReconstructedCV*[ ])
2: if (*ReconstructedCV*[*i*]= *CV*)
3: return(True);
4: end if
5: end For
6: return(False);

Algorithm 10. *VerifyResult* (Part 1)

INPUT: *ReconstructedCV*[ ], $\sigma_{k,name_j}$[ ][ ], *Names*[ ]
OUTPUT: A boolean value
//Calculation of the product of the reconstructed counter-values (*ReconstructedCV*[ ])
1: Prod_CV= 1;
2: For i ← 1 to length(*ReconstructedCV*[ ])
3: Prod_Cvj ← Prod_Cvj · *ReconstructedCV*[*i*];
4: end For

Algorithm 11. *VerifyResult* (Part 2)

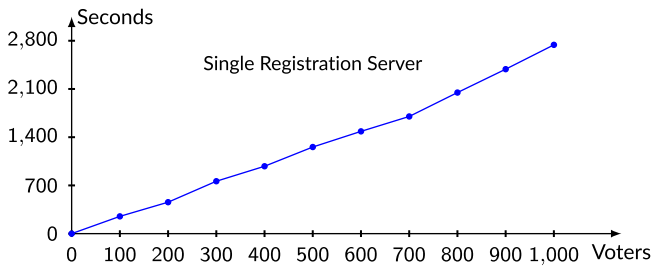
//Calculation of $\prod_{k=1}^{m}\prod_{j=1}^{m} e(Q_{name_j}, \sigma_{k,name_j})$
5: Prod_Pairing ← 1
6: For k ← 1 to *m*7: For j ← 1 to *m*
8: Prod_Pairing ← Prod_Pairing · pairing($H(Names[j])$, $\sigma_{k,name_j}[k][j]$);
9: end For
10: end For
11: If (Prod_Cvj = Prod_Pairing)
12: return(True);
13: else
14: return(False);
15: end If

5 | VYV PERFORMANCE EVALUATION

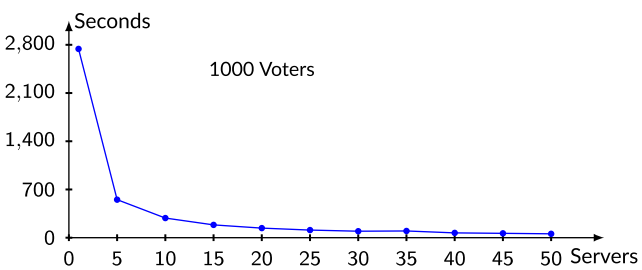
After implementing our protocol, we proceed to the evaluation of its performance. We vary *the number of voters, candidates and servers*, and measure, for each value of these parameters, the time required to execute each phase of our system. When we dispose of more than one server per authority to run the election, we use a load balancing strategy that allows us to ameliorate performance and exploit all the resources we have. Using the obtained values, we generate curves that show the variation of execution times as a function of these parameters. The obtained curves can be divided into two categories: (i) strictly monotonic curves (either increasing, when representing the variations of time as a function of voters number, or decreasing when representing the variations of time as a function of servers number) and (ii) increasing curves with constant parts (when representing the variation of the time as a function of candidates number, which is equal to the number of TAs). These constant parts are obtained when the servers number is not multiple of candidates number and the minimum number of servers per TA is the same for close values of TAs number. For each experimental value, we repeat the execution *100 times* and take the average of the 100 values obtained. We test our system with a number of voters ranging from *0 to 1000*, a number of candidates that varies between *1 and 31* and on *a single computer* (whose characteristics are mentioned in Subsection 4.3). Then, we will take two examples of real world elections, which are the last presidential election of Tunisia (2019) and that of France (2017), and we calculate the time taken by these elections if they were executed on our system. We mention here that we only evaluate the complexity of the cryptographic primitives without considering the time it takes to send the data via the blockchain.

5.1 | Registration time evaluation

As we mentioned before, the registration phase is off-line (and of course off-chain). Every voter moves to the nearest polling station to register and gets his/her authentication parameters. The total time that takes this phase depends only on the number of eligible voters and the number of servers that we dispose to run our system on. When running our system on a single server and with different values of voters number we get the curve represented in Figure 8A. To evaluate the speed-up when varying the number of servers, we fix the number of voters to *1000* and measure the time that takes our system to register these voters. Figure 8B shows the variation of this time as a function of the number of servers.



(A) Registration Time as a function of Voters Nbr.



(B) Registration Time as a function of Servers Nbr.

FIGURE 8 Evaluation of the registration time

5.2 | Setup time evaluation

During the setup phase, the election administrator generates all ballot numbers and their corresponding offsets. Then, TAs calculate countervalues. The total time that takes this phase depends on the number of ballots to generate, which is equal to the number of eligible registered voters (Figure 9A), the number of servers that we dispose to run this phase (Figure 9B), and the number of candidates of the election (Figure 9C). For each value of servers number of the curve represented by Figure 9B, we use a load balancing strategy to distribute the work of each TA on the different servers that it disposes.

5.3 | Authentication time evaluation

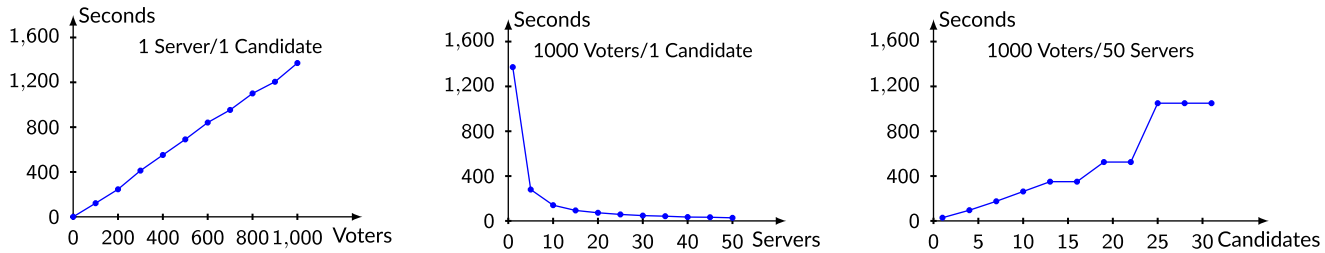
The majority of existing e-voting systems that include an authentication phase, verify each voter's credentials by checking their existence in a list that contains all registered voters' credentials. This verification has a quadratic complexity. In our case, the authentication phase has a linear complexity since we verify the validity of each voter's authentication parameters by executing only one multiplication operation. We evaluate the time taken by this phase as a function of voters number (Figure 10A) and servers number (Figure 10A). Candidates number has no influence on the authentication time.

5.4 | Voting time evaluation

During the voting phase, TAs encrypt and send a ballot to each eligible voter, who decrypts it, chooses his/her favorite candidate, encrypts his/her vote, and sends it back to TAs. The total voting time depends on the total number of ballots to encrypt (which is equal to the number of voters) and the number of servers. This variation is shown by Figure 11.

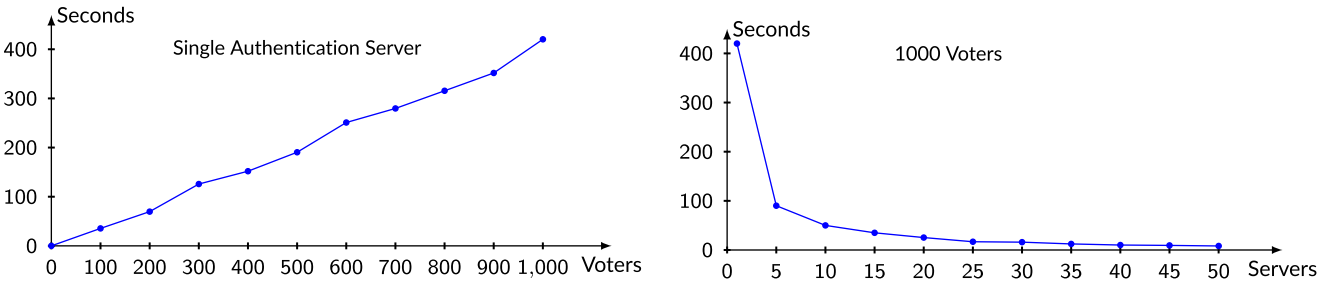
5.5 | Tallying time evaluation

The total time of the tallying phase depends on the number of votes, the number of servers, the way how voters voted, and the number of TAs (which is equal to the number of candidates) (Figure 12). In fact, each TA is dedicated to calculate the number of votes for a specific pseudo-ID. The best



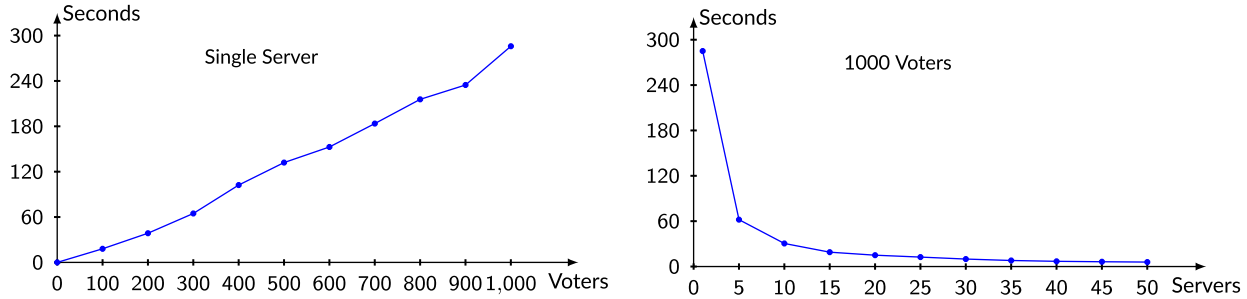
(A) Set up Time as a function of Voters Nbr. (B) Set up Time as a function of Servers Nbr. (C) Set up Time as a function of Candidates Nbr.

FIGURE 9 Evaluation of the setup time



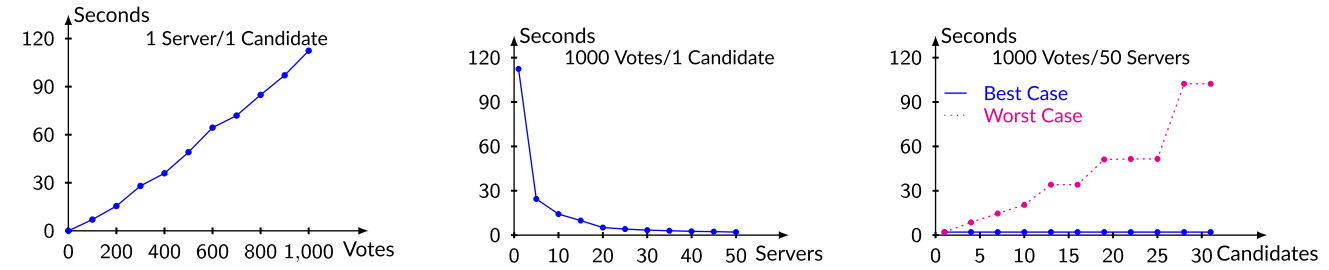
(A) Authentication Time as a function of Voters Nbr. (B) Authentication Time as a function of Servers Nbr.

FIGURE 10 Evaluation of the authentication time



(A) Voting Time as a function of Voters Nbr. (B) Voting Time as a function of Servers Nbr.

FIGURE 11 Evaluation of the voting time



(A) Tallying Time as a function of Voters Nbr. (B) Tallying Time as a function of Servers Nbr. (C) Tallying Time as a function of Candidates Nbr.

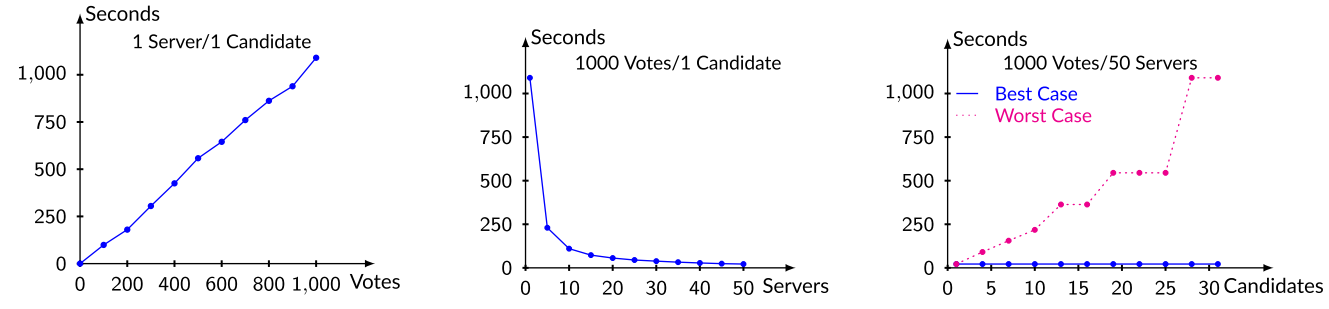
FIGURE 12 Evaluation of the tallying time

case is when TAs count an equal number of votes. The worst case is when only one TA counts all votes (all votes are for a unique pseudo-ID). When each TA disposes of more than one server to tally votes, it uses a load balancing strategy to distribute its work and exploit all servers it has.

5.6 | Verification time evaluation

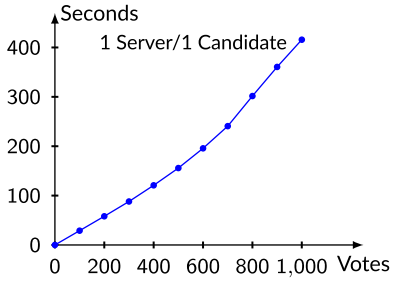
The verification phase is optional and includes:

- The reconstruction of voters receipts (by the TAs). The time of this step depends on the number of votes (Figure 13A), the number of servers (Figure 13B), the number of TAs (or candidates), and the way how voters voted (Figure 13C).
- The verification of the existence of a voter's countervalue in the reconstructed list. This step is performed by each voter who wants to verify that his/her vote was taken into account correctly, on his own computer. The time that takes this verification is about 0.07 second.
- The verification of Equation (1) to check the correctness of the final tally. The time that takes this step depends on the number of votes, the number of servers that dispose the verifier to check the equation, and the number of candidates. Figure 14 shows these variations.

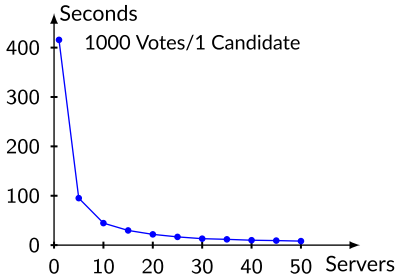


(A) Recons. of CV Time as a fct of Voters Nbr. (B) Recons. of CV Time as a fct of Servers Nbr. (C) Recons. of CV Time as a fct of Candidates Nbr.

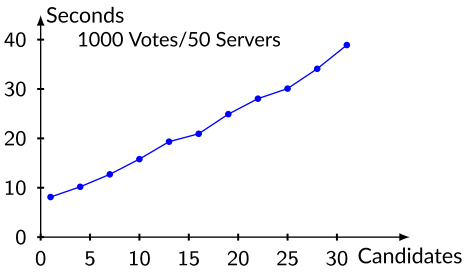
FIGURE 13 Evaluation of the reconstruction of countervalues time



(A) Univ. Verif. Time as a fct of Voters Nbr.



(B) Univ. Verif. Time as a fct of Servers Nbr.



(C) Univ. Verif. Time as a fct of Candidates Nbr.

FIGURE 14 Evaluation of the universal verification time

TABLE 1 Statistics of presidential Tunisian election of 2019

Number of registered voters	Number of voters first round	Number of candidates first round	Number of voters second round	Number of candidates second round	Number of polling stations
7 074 566	3 465 184	26	3 892 085	2	4567

TABLE 2 Execution of presidential Tunisian election of 2019 with VYV voting system

Number of servers	Setup time	Registration time	Authentication time 1st rnd / 2nd rnd	Voting time 1st rnd / 2nd rnd	Tallying time 1st rnd / 2nd rnd	Verification time 1st rnd / 2nd rnd
4567	11 h 5 5min 38 s	1 h 7 min 23 s	5 min 3 s/5 min 40 s	3 min 9 s/3 min 32 s	1 min 17 s/1 min 27 s	32 min 14 s/19 min 52 s

TABLE 3 Statistics of Presidential French election of 2017

Number of registered voters	Number of voters first round	Number of candidates first round	Number of voters first round	Number of candidates first round	Number of cantons
47 582 183	37 003 728	11	35 467 327	2	2054

TABLE 4 Execution of Presidential French Election of 2017 with VYV voting system

Number of servers	Setup time 1st rnd / 2nd rnd	Registration time	Authentication time 1st rnd / 2nd rnd	Voting time 1st rnd / 2nd rnd	Tallying time 1st rnd / 2nd rnd	Verification time 1st rnd / 2nd rnd
2054	3 days 4 h 15 min 12 s/ 15 h 32 min 27 s	16 h 47 min 42 s	2 h 6 s/1 h 55 min 6 s	1 h 14 min 45 s/ 1 h 11 min 39 s	30 min 43 s/29 min 26 s	4 h 18 min 4 s/ 1 h 57 min 51 s

5.7 | Example 1: Presidential election of Tunisia 2019

In this part, we evaluate the performance of our protocol in a large scale context. We take the example of the presidential Tunisian election of 2019. The statistics of this election are given in Table 1[10] and we give the time that takes each phase of our protocol to execute the Tunisian election in Table 2. The calculation is done based on the experimental measurements presented in the previous subsections. To execute our system, we suppose that we have a number of servers equal to the number of polling stations of the election.

5.8 | Example 2: Presidential election of France 2017

We take another example of an election with a larger number of voters which is the last Presidential election of France (2017). The statistics of this election are represented in Table 3[11]. We consider that we have a number of servers equal to the number of cantons in France to execute this election on our voting system. The results are given in Table 4.

[10] http://www.isie.tn/
[11] https://www.insee.fr/fr/statistiques/3540007

6 | DISCUSSION

We elaborate an informal security evaluation of VYV protocol and compare it with the security of the e-voting protocols studied in the related work section (see Section 2). We also give a formal proof for three security properties, which are voters authentication, votes secrecy, and votes privacy.

6.1 | Informal security evaluation of VYV

We evaluate and compare the security of VYV protocol with that of a smart contract for boardroom voting with maximum voter privacy (OVN), platform-independent secure blockchain-based voting system (PSBVS), an end-to-end voting-system based on bitcoin (EtEVBB), end-to-end voting with nonpermissioned and permissioned ledgers (EtEVnPPL), follow my vote (FMV), and TIVI. The summary of the comparison is given in Table 5.

- *Eligibility: VYV* protocol guarantees that only eligible voters join the election blockchain and participate to the voting process since registration authorities verify each voter identity via a face to face meeting, during the registration phase, and only eligible voters are provided with authentication parameters. The list of eligible registered voters is then published on the blockchain and auditable by everyone. Our protocol also includes an authentication phase in which we verify the validity of the authentication parameters entered by the voter. In the case of *OVN* and *FMV*, we need to trust, respectively, the election administrator and a centralized authority to guarantee this property because they are the only ones responsible of authenticating voters. *PSBVS* does not ensure, also, voter's eligibility since it does not verify, physically or by using biometric techniques, the eligibility of the voter. A voter can register him/herself by simply providing his/her e-mail address, identity number, or an invitation URL with a desired password. *TIVI* respects this property since it checks the elector's identity via a selfie and by using the facial recognition technology. EtEVBB uses an anonymous Kerberos authentication protocol to authenticate voters and EtEVnPPL uses a dedicated private blockchain to manage voters identities. In both cases, we need to trust an authority to ensure voters eligibility.
- *Individual verifiability: VYV* protocol gives the possibility to each eligible voter to check that his/her vote is cast as intended by checking the existence of his countervalue in the list of reconstructed countervalues, published during the verification phase. Due to the transparency of blockchain, in all other voting systems each eligible voter has the possibility to verify the presence and the correctness of his/her vote in the ballot box by inspecting the blockchain and verifying the existence of a transaction containing his/her vote.
- *Universal verifiability:* In the case of *VYV*, voters check the accuracy of the final result by checking Equation (1), which verifies the equality between the product of reconstructed counter values and the sum of the counts displayed by tallying authorities. *OVN* is a self-tallying protocol; thus, it ensures universal verifiability. *TIVI* ensures universal verifiability by using blockchain technology. Voters in *PSBVS* check the correctness of the election final result by comparing the number of eligible voters with the number of recorded and counted ballots. In EtEVBB and EtEVnPPL, voters verify the accuracy of the election final result by inspecting the blockchain and summing tokens received by each candidate. However, *FMV* is not universally verifiable.
- *Vote-privacy:* In the case of *VYV*, we cannot make a link between a voter and his/her vote included in the final tally, since votes are cast encrypted and recorded using blockchain technology. Like *VYV*, *OVN*, *TIVI*, EtEVBB, and EtEVnPPL ensure vote-privacy, by using blockchain technology.

TABLE 5 Security evaluation of VYV, OVN, TIVI, FMV, PSBVS, EtEVBB, and EtEVnPPL, where multiple or single indicates the number of possible votes

	VYV	OVN	TIVI	FMV	PSBVS	EtEVBB	EtEVnPPL
Eligibility	✓	Trusted admin	✓	Trusted authority	×	Trusted authority	Trusted authority
Individual verifiability	✓	✓	✓	✓	✓	✓	✓
Universal verifiability	✓	✓	✓	×	✓	✓	✓
Vote-privacy	✓	✓	✓	Trusted authority	✓	✓	✓
Receipt-freeness	✓	×	×	×	✓	×	✓
Coercion resistance	×	×	×	×	×	×	×
Fairness	✓	×	✓	×	✓	×	Trusted authority
Integrity	✓	✓	✓	×	✓	✓	✓
Robustness	✓	×	✓	✓	✓	✓	✓
Vote-and-go	✓	×	✓	✓	✓	✓	✓
Voting policy	Multiple	Single	Single	Multiple	Single	Single	Single

TABLE 6 ProVerif results and execution times

Properties	Description	Query	Result	Time (s)
Vote secrecy	To capture the value of a given vote, an attacker has to intercept the values of two parameters: the ballot number BN and the pseudo ID of the chosen candidate Cj.	*Query attacker (BN) Query attacker (Cj)*	Proved	0.019
Voter's Authentication	We use correspondence assertion to prove this property.	*Event accepted Authentication*(S_{PW}, P_{PW}) ⇒ Event Verifies Parameters(SPW, PPW)	Proved	0.017
Vote privacy	To express vote privacy we prove the observational equivalence between two instances of our process that differ only in the choice of candidates.	*VYV*(SK_{V1},*choice*[*V1*, *V2*])| VYV(SKV2,choice[V2, V1]).	Proved	0.038

However, *FMV* needs to trust a centralized authority to hide the correspondence between the voters' real identities and their voting keys. Finally, *PSBVS* ensure votes privacy due to the use of blockchain technology and the short linkable ring signature (SLRS).

- *Receipt-freeness:* Our protocol ensures receipt freeness due to our ballot structure that includes countervalues $CV_{BN,name_j,k}$ that allow voters to verify their votes without disclosing the chosen candidates. On the other hand, *OVN*, *TIVI*, *FMV*, and EtEVBB do not provide voters with receipts that allow them to verify their votes without revealing the value of the vote. In *PSBVS*, voters verify the existence of their votes in the ballot box by inspecting the blockchain and checking the existence of transactions bearing their signatures. EtEVnPPL respects this property thanks to the use of Multichain platform.
- *Coercion resistance:* With reference to Juels et al. definition of coercion resistance,[6] all the studied systems, including our proposed protocol, are not coercion resistant. A coercer can force a voter to vote for a certain candidate and check his submission later.
- *Fairness:* This property is ensured by *VYV*, *TIVI*, and *PSBVS* due to the encryption of votes before being cast; thus, nobody can get partial results before the end of the voting phase. EtEVnPPL also ensures fairness due to the use of private blockchain. However, in *OVN*, voters who have not yet cast a vote, can cooperate and calculate a partial result. Also, *FMV* does not ensure fairness because votes are cast without being encrypted. Finally, EtEVBB compromise this security property since voters can watch the election progress in real time.
- *Integrity:* VYV protocol and all other systems use the blockchain technology as a ballot box. Blockchain is characterized by the immutability of its transactions. Thus, they ensure the integrity of the stored data, with exception of *FMV*, which gives the possibility to a centralized authority to change votes by using the voters' pass-phrases.
- *Robustness:* Except *OVN*, in which a dishonest voter can invalidate the election by refusing to cast a vote for example, all the other voting systems, including our proposed solution, resist to the misbehavior of dishonest voters.
- *Vote-and-go:* Unlike *OVN* which is a self-tallying voting protocol, VYV and the other voting systems do not need the voter to trigger the tallying phase, they can cast their votes and quit before the voting ends.
- *Voting policy: VYV* and *FMV* give the possibility to eligible voters to vote more than once and only their last votes are counted. However, *OVN*, *TIVI PSBVS*, EtEVBB, and EtEVnPPL record and save only the first valid vote of each eligible voter and discard the other ones.

6.2 | Formal security evaluation of VYV

ProVerif[20] is an automatic symbolic protocol verifier, capable of proving reachability properties, correspondence assertions, and observational equivalence[21] of security protocols. To perform an automated security analysis using this verification tool, we model our protocol in the Applied Pi-Calculus.[22] This modeling language is a variant of the Pi-Calculus extended with equational theory over terms and functions and provides an intuitive syntax for studying concurrency and process interaction. The Applied Pi-Calculus allows us to describe several security goals and to determine whether the protocol meets these goals or not. We use the classical intruder model and the standard modeling of the security properties proposed by Dreier et al.[1] in our ProVerif code.

Because of the limitation on the number of pages, we put all ProVerif codes online[12] and give the queries, the results of executing these codes, and the time it takes ProVerif to prove the properties in Table 6.

[12] http://sancy.univ-bpclermont.fr/%7Elafourcade/ProverifVYV.tar

7 | CONCLUSION

In this article, we have proposed a secure online electronic voting protocol based on a variety of cryptographic primitives, namely, elliptic curve cryptography, identity-based encryption, pairing, and Paillier cryptosystem. Called Verify-Your-Vote,[2] this protocol uses Ethereum blockchain as a public bulletin board. We have implemented this protocol and evaluated its performance in terms of time, cost, and the number of voters and candidates that can be supported. Our evaluation considers only the cost related to the voting protocol itself, which is more significant than the one related to blockchain operations. We have proved that VYV protocol ensures voters eligibility, individual and universal verifiability, votes secrecy, voters privacy, receipt freeness, fairness, and robustness. We have also elaborate a formal proof of three security properties, using the Applied Pi-Calculus modeling language, and ProVerif tool. However, VYV protocol does not ensure coercion resistance and should be executed only in a low coercive environment. Thus, future work will be dedicated to ameliorate the security of this protocol to ensure coercion resistance.

ORCID

Marwa Chaieb https://orcid.org/0000-0003-3757-8650

REFERENCES

1. Dreier J, Lafourcade P, Lakhnech Y. A formal Taxonomy of privacy in voting protocols. Paper presented at: Proceedings of the 2012 IEEE International Conference on Communications (ICC); 2012:6710-6715; IEEE.
2. Chaieb M, Yousfi S, Lafourcade P, Robbana R. Verify-your-vote: a verifiable blockchain-based online voting protocol. In: Marinos T, Rupino CP, eds. *Information Systems - 15th European, Mediterranean, and Middle Eastern Conference, EMCIS 2018, Limassol, Cyprus, October 4-5, 2018, Proceedings, Lecture Notes in Business Information Processing*. Vol 341. Limassol, Cyprus: Springer; 2018:16-30.
3. McCorry P, Shahandashti SF, Hao F. A smart contract for boardroom voting with maximum voter privacy. In: Aggelos K, ed. *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers, Lecture Notes in Computer Science*. Vol 10322. Sliema, Malta: Springer; 2017:357-375.
4. Yu B, Liu JK, Sakzad A, et al. Platform-independent secure blockchain-based voting system. In: Chen L, Manulis M, Schneider S, eds. *Information Security - 21st International Conference, ISC 2018, Guildford, UK, September 9-12, 2018, Proceedings, Lecture Notes in Computer Science*. Vol 11060. Guildford, UK: Springer; 2018:369-386.
5. Paillier P. Paillier encryption and signature schemes. In: Tilborg HCA, Jajodia S, eds. *Encyclopedia of Cryptography and Security*. 2nd ed. New York, NY: Springer; 2011:902-903.
6. Juels A, Catalano D, Jakobsson M. Coercion-resistant electronic elections. In: Atluri V, De Capitani VS, Dingledine R, eds. *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005*. Alexandria, VA: ACM; 2005:61-70.
7. Vukolic M. The quest for scalable blockchain fabric: proof-of-work vs. BFT replication. In: Camenisch J, Kesdogan D, eds. *Open Problems in Network Security - IFIP WG 11.4 International Workshop, iNetSec 2015, Revised Selected Papers, Lecture Notes in Computer Science*. Vol 9591. Zurich, Switzerland: Springer; 2015:112-125.
8. Dinh TTA, Wang J, Chen G, Liu R, Ooi BC, Tan K-L. BLOCKBENCH: a framework for analyzing private blockchains. In: Salihoglu S, Zhou W, Chirkova R, Yang J, Suciu D, eds. *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017*. Chicago, IL: ACM; 2017:1085-1100.
9. Bistarelli S, Mantilacci M, Santancini P, Santini F. An end-to-end voting-system based on bitcoin. In: Seffah A, Penzenstadler B, Alves C, Peng X, eds. *Proceedings of the Symposium on Applied Computing, SAC 2017*. Marrakech, Morocco: ACM; 2017:1836-1841.
10. Steiner JG, Neuman B, Clifford SJI. Kerberos: an authentication service for open network systems. *Usenix Winter*. Dallas, TX: USENIX Association; 1988:191-202.
11. Zhu L, Leach PJ, Hartman S, Emery S. Anonymity support for Kerberos. *RFC*. Vol 8062; 2017(1):1-18. https://doi.org/10.17487/RFC8062.
12. Bistarelli S, Mercanti I, Santancini P, Santini F. End-to-end voting with non-permissioned and permissioned ledgers. *J Grid Comput*. 2019;17(1):97-118. https://doi.org/10.1007/s10723-019-09478-y.
13. Koblitz N. Constructing elliptic curve cryptosystems in characteristic 2. In: Menezes A, Vanstone SA, eds. *Advances in Cryptology – CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings, Lecture Notes in Computer Science*. Vol 537. California, CA: Springer; 1990:156-167.
14. Boneh D. Pairing-based cryptography: past, present, and future. In: Wang X, Sako K, eds. *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, December 2-6, 2012. Proceedings, Lecture Notes in Computer Science*. Vol 7658. Beijing, China: Springer; 2012:1.
15. Shamir A. Identity-based cryptosystems and signature schemes. In: Blakley GR, David C, eds. *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings, Lecture Notes in Computer Science*. Vol 196. California, CA: Springer; 1984:47-53.
16. Boneh D, Franklin MK. Identity-based encryption from the weil pairing. In: Kilian J, ed. *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings, Lecture Notes in Computer Science*. Vol 2139. California, CA: Springer; 2001:213-229.
17. Gennaro R, Jarecki S, Krawczyk H, Rabin T. Secure distributed key generation for discrete-log based cryptosystems. In: Stern J, ed. *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding, Lecture Notes in Computer Science*. Vol 1592. Prague, Czech: Springer; 1999:295-310.
18. Paillier P. Public-key cryptosystems based on composite degree residuosity classes. In: Jacques S, ed. *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding, Lecture Notes in Computer Science*. Vol 1592. Prague, Czech: Springer; 1999:223-238.
19. Akinyele JA, Garman C, Miers I, et al. Charm: a framework for rapidly prototyping cryptosystems. *J Cryptograph Eng*. 2013;3(2):111-128. https://doi.org/10.1007/s13389-013-0057-3.

20. Blanchet B. Automatic verification of security protocols in the symbolic model: the verifier proverif. In: Aldini A, López J, Martinelli F, eds. *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures, Lecture Notes in Computer Science*. Vol 8604. New York, NY: Springer; 2013:54-87.
21. Delaune S, Kremer S, Ryan M. Verifying privacy-type properties of electronic voting protocols. *J Comput Sec*. 2009;17(4):435-487. https://doi.org/10.3233/JCS-2009-0340.
22. Abadi Martín, Blanchet B, Fournet C. The applied pi calculus: mobile values, new names, and secure communication. *J ACM*. 2018;65(1):1:1-1:41. https://doi.org/10.1145/3127586.

How to cite this article: Chaieb M, Yousfi S, Lafourcade P, Robbana R. Design and practical implementation of verify-your-vote protocol. *Concurrency Computat Pract Exper*. 2020;e5813. https://doi.org/10.1002/cpe.5813