Homomorphic Encryption in E-Voting Systems: The University of Jordan Case Study

Mohammed Arabiat*, Nael Al-Basheer†, Khair Eddin Sabri‡ and Hazem Hiary§
Computer Science Department
King Abdullah II School for Information Technology
The University of Jordan, Amman, Jordan
*Email: mohammed.arabiat@outlook.com
†Email: nael.albasheer@outlook.com
‡Email: k.sabri@ju.edu.jo
§Email: hazemh@ju.edu.jo

Abstract—Homomorphic encryption has been explored theoretically for use in electronic voting systems. One of the well known techniques is voting based on Paillier encryption, though few applications or studies of the practicality of its uses have been done. In this paper, we discuss the applicability of the Paillier homomorphic encryption in real world applications by taking the University of Jordan as a case study. We adapt the original technique to fit the University of Jordan voting requirements. We present the design and implementation of the proposed system. We eventually state the practicality of the system and the possible shortcomings. We also discuss the merits of the system in comparison with traditional voting methods. We do not contribute to the security of e-voting, our contribution is in the application of homomorphic encryption to an election as a case study.

Index Terms—Secure voting system, Homomorphic encryption, Software design

1 Introduction

VOTING is the most prominent facet of democracy. It is the mechanism with which people make decisions that affect them all. In ancient Athens, elections were conducted by counting the raised hands of people gathered in a square. Over the centuries, democracy and voting procedures have evolved to meet with the growing number of population. Today, elections are usually operated in the form of the secret ballot method using paper ballots and manual counting of votes. Governments have started using electronic ballot casting and tallying machines, though still these machines are far from what one would think of as an electronic voting system.

Fully electronic voting systems face a multitude of issues in legality, morality and transparency. One only needs to realize how easy it is to fake a number in a database compared to faking physical paper ballots to appreciate the distrust people have towards such systems. Electronic voting systems are also more opaque than the traditional methods. Not everyone understands computer and security concepts which reduces peoples' trust in those systems regardless of how trustworthy they might be.

Homomorphic cryptosystems have the unique property where operations on the cipher-text results in the operations being reflected on the original clear-text numbers after decryption. Pascal Paillier has developed a public key cryptosystem [1] that possesses such a property. This property has been noticed to be useful in electronic voting scenarios [2]. We apply these techniques and discover the feasibility of the system. We are interested in finding the merits and shortcomings of such an electronic voting scheme in a real-world setting. We do not improve on the security of the methods utilized. We apply the aforementioned studies to design and implement a secure electronic voting system for the University of Jordan.

The scope of this work is designing and implementing an e-voting system using homomorphic encryption based on Damgård and Jurik's work [2] and to evaluate their efficiency and effectiveness.

We evaluate multiple characteristics of the system. Namely, security, performance and accessibility. We are also interested in other features but at a lower priority such as uncoercibility and participation anonymity [3].

This project is not breaking new grounds in terms of secure voting schemes. Nor is it designing new rules for the elections in the University of Jordan. We do not consider student authentication, as we assume such a system already exists.

2 Literature Review

In this section we give the history of Paillier's cryptosystem and the subsequent work that has gone into it. We also discuss the works related to electronic voting in general which have detailed the requirements needed for a system to be considered a legal voting mechanism.

2.1 Paillier's Cryptosystem

In 1999, Pascal Paillier published a paper [1] detailing a public key encryption method which had the property of being *homomorphic*. Such a property is defined as being able to process data in its encrypted form so that when

we decrypt the results it will be as though the operations have been carried out on the data in its plain-text form. The cryptosystem derives its security from the difficulty of deciding composite residuosity.

Afterwards, in 2001, Ivan Damgård and Mads Jurik have generalized and simplified Paillier's original cryptosystem [2]. They have generalised the cryptosystem where Paillier's algorithm is a variation of it. They have also provided a thresholded version of the Paillier cryptosystem [4].

2.2 Electronic Voting

It is easy to see the benefits of a well designed electronic voting system. It would provide us with conveniences that we cannot have in a traditional system. Though designing a system that can replace traditional voting is a difficult and complicated task as the system has to account for many legal and moral regulations. These have been summarized in a paper published by Costas Lambrinoudakis and his colleagues [3]. The paper has also summarized various proposed techniques for electronic voting and has evaluated them on the basis of the requirements it has detailed. We defer further review to the aforementioned paper.

Most of the presented voting systems do not analyse the implementation of the system such as [5], [6], [7]. The most related work to ours is the Civtas voting system [8]. However, Civtas is based on El Gamal encryption while our system is based on Paillier.

3 Proposed Architecture

In this section we describe the original architecture by Damgård and Jurik [2] and then we describe our adaptations to the architecture to make it applicable for use in the University of Jordan.

3.1 Damgård and Jurik's Original Architecture

This method relies on Damgård and Jurik's own thresholded version of Paillier's cryptosystem [2].

In their system, a *yes/no* poll is achieved by first generating a public encryption key and a private decryption key. The public key is available to everyone while the private key is held by the authorities. A voter encrypts their ballot consisting of a number either zero or one. The general idea is that when summing up the ballots the result is the number of *yes* votes and the number of overall votes is known. So it is easy to compute the number of *no* votes as well.

The *yes/no* system can be extended into a *1-out-of-L* election by holding L parallel *yes/no* polls. In this system the voter encrypts L numbers and each number will be accounted for one of the L candidates. In an election where a voter can at most vote for one candidate, their ballot must contain at most one value containing 1 and the rest be zeroes.

It is important to note that for each number a value r is generated randomly and it is used by the encryption algorithm to make the encrypted content differ every time. Therefore, a 1 or a 0's ciphertext is different each time a different r is used.

Damgård and Jurik have also provided interactive proof methods that would enable anyone to verify any vote to be valid in that it only contains the value 0 or 1 without exposing the actual value and to verify there are no malicious users voting for more than the maximum allotted number of votes they are allowed. In the literature, this number is denoted t. The proof is simple. By providing $\prod_{i=1}^{L}(r_i) \bmod n^{s+1}$. This allows the sum of their votes to be revealed in an encrypted form verifiable to be equal to t.

Once the votes have been collected, the votes are added up and the plain-text of the result can be reconstructed from the output of the authorities as they decrypt the final tally in the thresholded scheme. Considering the thresholded nature of the algorithm, the tally is trusted to be correct unless the number of malicious authorities exceeds a certain threshold.

3.2 Adaptations of the Original Architecture

We first justify the adaptations proposed to the system by describing the election process in the University of Jordan. Every student has the right to vote for a candidate from their own electoral district (usually the academic department their major is in) and the right to vote for one of many lists of candidates at the scale of the university.

We see that we have restrictions on each student on which individual candidate they vote in. Such a restriction is not accounted for in the original paper. We can also see that while the restriction applies on the vote for an individual candidate, it does not exist for the university-wide vote for a list of candidates.

We intend to solve that issue by having $Q + 1$ parallel *1-out-of-L* elections where Q is the number of electoral districts in the university. The extra election is for the university-scale lists election. Each voter is entitled to vote in one of the Q individual candidate elections which concerns the electoral district the voter belongs to and in the university-wide lists election.

For each election of the $Q + 1$ elections, a public encryption key and a private decryption key are generated. The public keys are available publicly while the private keys are held by the authority. We use the original, non-thresholded Paillier cryptosystem.

Each voter encrypts L numbers where 1 of them contains the plain-text value 1 (t is equal to 1 for the purposes of voting in the University of Jordan) for each ballot. They also include the honest verifier zero-knowledge proofs.

A voter constructs two ballots. One for the election of the level of the electoral district and another for the election on the level of the university district each encrypted with the respective public key.

Ballot submission does not differ from the original method but tallying does. Considering the authority has access to the private decryption key, it can recover the original plain-text of each vote in any given ballot. Afterwards, finding the original r is possible. As we now have all the values but r in the encryption equation $E(i, r) = g^i r^n \bmod n^2$ [1] where i is the plain-text and r is the random number. The authority would post the final count with $\prod_{i=1}(r_{ji})$ for all $j \in \{1, ..., L\}$ where a product of rs is computed for each candidate. This allows everyone to encrypt the final tally with r being the product given by the authority and compare it to their own result of the tally.

4 Requirement Engineering

4.1 Introduction

Here we detail the requirements of the system in software engineering terms. We will not list all requirements in this paper, just the ones that most pertain to the design.

The election has been categorized into different *phases* [3]. These phases are as follows:

4.1.1 Voter registration

The phase where eligible voters are registered for voting.

4.1.2 Voter authorization

The phase where a voter proves their identity to be able to vote.

4.1.3 Vote casting

The phase where votes are submitted to the system. Might include the process of validating the ballots as they come in.

4.1.4 Vote tallying

The phase where the votes are counted and a final result is reached.

4.1.5 Tally verification

The phase where the final result is audited to confirm its validity

Tally verification is the only phase not directly related to the absolute core functionality of a voting system. It is a step that is not mandatory unless the validity of the results is questioned.

4.2 Legal Requirements

Eliciting requirements for e-voting is a process of multiple facets, mainly the legal aspect and the traditional software engineering point of view. It is important to lay out the legal issues with e-voting, as they affect the functional and non-functional requirements of a system very deeply.

Legal requirements for e-voting can be summarized in the following points [9]:

4.2.1 Generality

The requirement that all eligible voters are able to vote.

4.2.2 Freedom

The requirement that a voter shall have full control of their vote.

4.2.3 Equality

The requirement that all voters have an equal effect on the outcome of the election.

4.2.4 Secrecy

The requirement that a voter shall be confident in the fact that their vote cannot be linked back to them.

4.2.5 Directness

The requirement that there shall be a lack of intermediaries in the voting procedure. Therefore, each and every ballot affects the procedure directly.

4.2.6 Democracy

The requirement that only eligible voters are allowed to vote and that a voter can only vote once.

It is worth noting that these requirements contradict at some junctions. For example, *secrecy* contradicts with *democracy* in respect to verifiability. As a voter cannot verify their vote if *secrecy* is to be upheld.

These requirements are to be respected by our system as they reflect the general intention of the student union elections in the University of Jordan.

4.3 Functional Requirements

Functional requirements mostly deal with mapping e-voting concepts to equivalent traditional voting concepts. Specifying the roles that are expected from the system for its users to be able to practice their democratic rights. These requirements are as follows:

4.3.1 Automatic voter registration

The system shall be able to identify and list the eligible voters. An eligible voter is a student who is currently enrolled in an academic program in order to earn a degree. This requirement is directly related to the *voter registration* phase.

4.3.2 Automated voter authentication

A voter shall be able to prove their identity in order to gain access to vote casting facilities. An authenticated voter should only vote once and the authentication can be used to enforce this restriction. The *voter authorization* phase is related to this requirement.

4.3.3 Electronic vote casting

The system shall provide the means necessary for voters to cast their votes. This requirement directly deals with the *vote casting* phase.

4.3.4 Automatic vote tallying

At the time of the *vote tallying* phase, the operation shall be done automatically.

4.3.5 Automatic tally verification

Tally verification (if required) shall be carried out automatically and correctly. Tally verification should also be a transparent operation. This requirement is related to the *tally verification* phase.

4.3.6 Receipt freeness

Receipt freeness is the concept of proving to a voter that their vote has been counted without providing them with a meaningful receipt that can prove the voter's choice [10]. This is also known as uncoercibility.

4.3.7 Electoral district voting

In the *vote casting* phase, a voter shall be able to vote for a single candidate from the electoral district the voter belongs to. A voter shall not be able to vote for more than one candidate nor shall they be able to vote for a candidate from another electoral district.

4.3.8 University-wide list voting

In the *vote casting* phase, a voter shall be able to vote for a list of candidates from multiple electoral districts. A voter shall not be able to vote for more than one list of candidates. No electoral district restrictions apply.

4.3.9 Fairness

The outcome of the election shall only be known after the end of the election. This requires the separation of tallying from the *vote casting* phase.

4.3.10 Accuracy

The announced tally shall exactly match the actual outcome of the election.

4.3.11 Walk-away property

A voter shall be able to cast their vote with no need to return for another round of communication with the election system [3].

4.4 Non-functional requirements

Non-functional requirements are not apparent to the end users but are very influential to the internal design and architecture of the system. They usually address the performance, reliability and security of a system. The non-functional requirements of the system are listed below.

4.4.1 Robustness

No reasonably sized coalition of voters or authorities or a third party can disrupt the election. This also includes preventing misbehavior of voters and authorities such that no party can claim the invalidity of the results due to a misbehavior of one of the actors. This also implies the system's ability to withstand external attacks such as DDoS [3].

This requirement is mostly concerned with the *vote casting* phase. As it is the most traffic-heavy phase in the election process.

4.4.2 Efficiency

The system shall be performant and responsive. This is most important in the *vote casting* phase but is also important in the *vote tallying* phase.

4.4.3 Security

The system shall be secure from all types of attacks. Starting from cryptanalysis attacks to the core algorithms in use up to software exploitation such as SQL injection, XSS or CSRF attacks. Voter-authority communication shall be encrypted, as shall be any authority-authority communication.

All interactions in the system utilize SSL/TLS encryption. While the votes are homomorphically encrypted, this is mainly to protect them at the authority's endpoints and when undergoing auditing. For security across the communication channels, we will use standard SSL/TLS encryption.

5 System Design

Continuing with the software engineering theme, we present the design of the system using design diagrams.

5.1 Context

A context diagram shows the relationship between a system and its neighboring systems. It is used to clearly define the boundaries of functionality of the system in question. In Figure 1, the voting system is shown to only interact with the registration system. It does so for voter authentication purposes (requirement 4.3.2) as we assume such a mechanism already exists in the current registration system. We encrypt all communication between the voter and the system.(requirement 4.4.3).

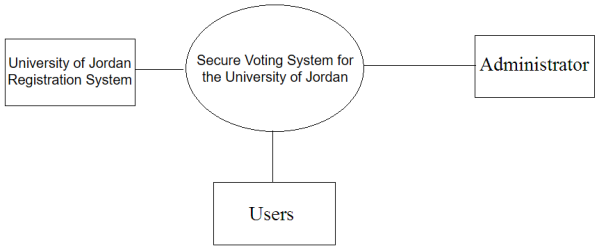

Fig. 1. System Context Diagram

5.2 Entity-Relationship

Figure 2 details the entities of the system and the relationships between them.

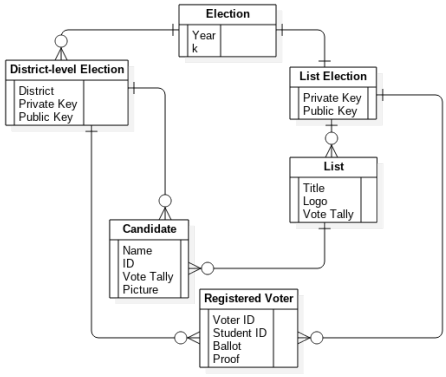

Fig. 2. System Entity-Relationship Diagram

An explanation of the entities is as follows.

5.2.1 Election

Defines a year's election. Contains one *District Level Election* (5.2.2) for each district (one-to-many relationship) and one *List Election* (5.2.3) on the level of the university (one-to-one relationship). An entity of this type is to be created for each year. Table 1 describes its attributes.

5.2.2 District-Level Election

A district-level election is an election held in the scope of an election district. Each one of these elections is held independently from the other with a disjoint set of *candidates* (5.2.5) and *registered voters* (5.2.6). Table 2 describes the attributes of this entity.

TABLE 1
Election Entity Attributes

Attribute	Description
Year	The year at which the election was held.
k	Paillier parameter k. Byte length of n.

TABLE 2
District-Level Election Entity Attributes

Attribute	Description
District	Contains the identifier of the district this entity represents.
Private Key	The private decryption key for the ballots.
Public Key	The public encryption key for the ballots.

5.2.3 List Election

A *List Election* entity represents a university-wide election where voters vote for a list of candidates that are not bound by the same electoral districts. The entity associates with *lists of candidates* (5.2.4) in a one-to-many relationship. Table 3 describes the attributes of this entity.

TABLE 3
List Election Entity Attributes

Attribute	Description
Private Key	The private decryption key for the ballots.
Public Key	The public encryption key for the ballots.

5.2.4 List

A *List* entity represents a list of candidates to be voted for in the election. It associates with all member *candidates* (5.2.5) in a one-to-many relationship and with a *list election* (5.2.3) in a many-to-one relationship. Table 4 describes its attributes.

TABLE 4
List Entity Attributes

Attribute	Description
Title	The title of the list. Their human-friendly identifier.
Logo	The logo of the list for easy discrimination by the voter.
Vote Tally	The amount of votes this list has garnered.

5.2.5 Candidate

A *candidate* is a student running for the student union. They meet the requirements and they are associated with exactly one district and at most one *list* (5.2.4) in a many-to-one relationship with partial participation. Table 5 details the attributes of this entity.

5.2.6 Registered Voter

This entity holds the information of a registered voter including their ballot. A registered voter is enumerated twice: Once for the list election and another for the voter's district election. This means each student at the university has control over two voter IDs and ballots. Table 6 details the attributes of this entity.

TABLE 5
Candidate Entity Attributes

Attribute	Description
Name	The name of the candidate.
ID	The student ID of the candidate.
Vote Tally	The amount of voters who voted for this candidate
Picture	A picture for easy discrimination by the voter.

TABLE 6
Registered Voter Entity Attributes

Attribute	Description
Voter ID	A unique ID generated for the voter.
Student ID	The student's ID.
Ballot	The ballot submitted by the voter.
Proof	The proof of validity of the ballot.

5.3 Use Cases

Use case diagrams are used to define the actors in the system and their roles. Fig. 3 illustrates the use cases of the system. Table 7 details the actors in the system. Table 8 describes the use cases.

Not all of these use cases are related to the security aspect, of course. But it is important to clearly define the actors and their roles in the system to be able to understand the operations that the system should implement. We will see the value of these definitions when we get to the interactions and their sequence.

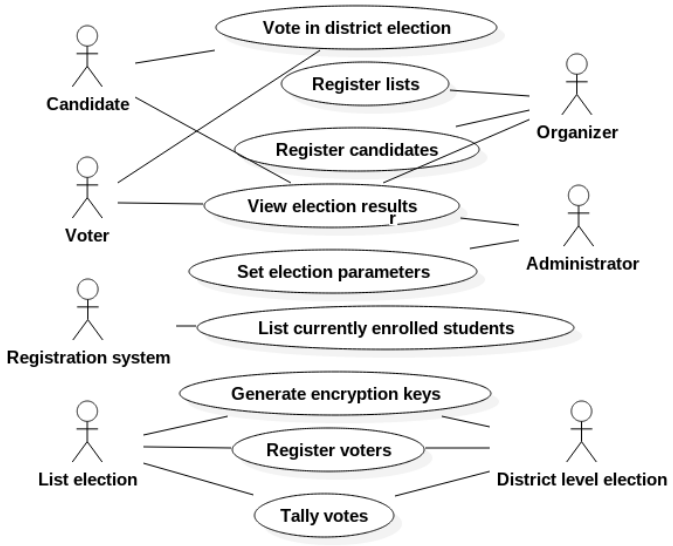

Fig. 3. System Use Case Diagram

5.4 Sequence

A sequence diagram deals with modeling the interaction between various actors in the system. The actors have been discussed in 5.3. We showcase two different interactions and their sequence of operations: *Authentication* and *vote submission*. All interactions are encrypted with SSL/TLS.

TABLE 7
Actors in the Voting System

Actor	Description
Candidate	A student registered to be voted for.
List	A list of candidates to be voted for across the university.
Voter	A student who votes for a candidate and a list.
Registration System	Pre-existing system for authentication.
List election	The system portion handling the list election.
District level election	The system portion handling the district elections.
Organizer	The person responsible for registering candidates.
Administrator	Person with administrative access to the system.

TABLE 8
Use Cases in the Voting System

Use Case	Description
Vote in District Election	Cast a vote for a candidate.
Register Lists	Enter the members and other details of lists.
Register candidates	Enter the details of candidates.
View election results	View the final tally after the election has ended.
Set election parameters	Set parameters pertaining to all aspects of the process.
List currently enrolled students	Find all enrolled students.
Register voters	Generate unique voter IDs for eligible voters.
Tally votes	Count the votes for each candidate or list.

5.4.1 Authentication

In Fig. 4, we illustrate the interaction between a voter, an election system and the registration system to achieve authentication for the voter.

In the first step, the student sends their credentials to election server. Afterwards, the election system verifies whether the student is registered with the system, if the verification fails, an error is raised (step 3). Otherwise, the system authenticates the user by forwarding the credentials to an existing authentication infrastructure (step 2). The registration system authenticates the student and return the result of the authentication (steps 4 and 6) to the voting system. If the result was failure, the failure is forwarded to the student (step 7). Otherwise, the voting system returns the election parameters to the user as they have been successfully authenticated (step 5).

5.4.2 Vote submission

The vote submission interaction is specified in Fig. 5.

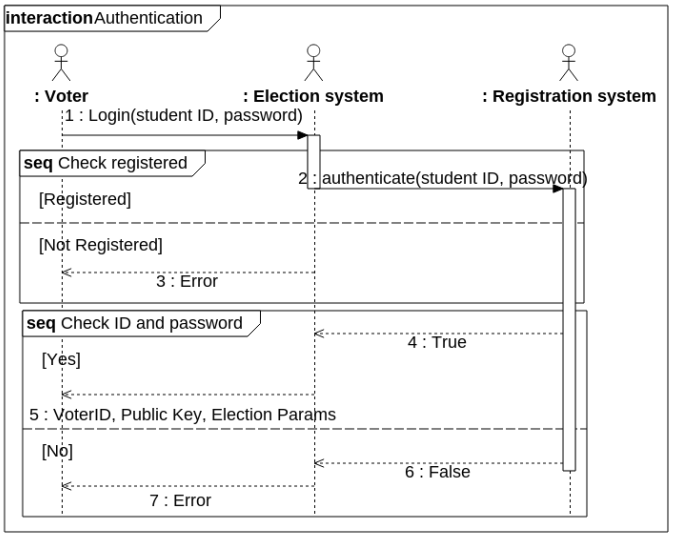

Fig. 4. Authentication Sequence Diagram

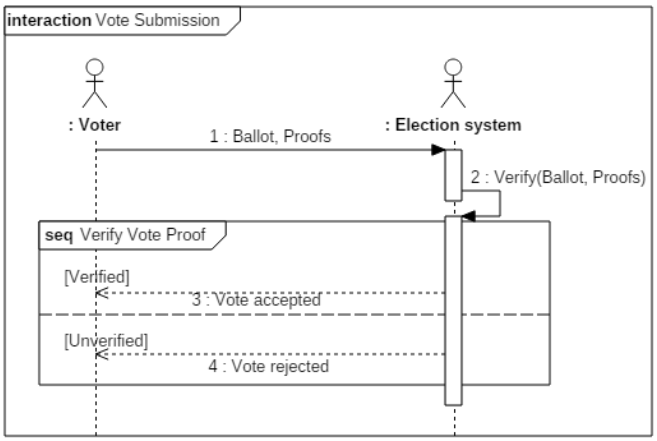

Fig. 5. Vote Submission Sequence Diagram

The voter sends their ballot and the proofs of validity (step 1). While this is considered one step in the diagram, it is an interactive protocol with multiple steps [2]. The system then verifies the ballot (step 2). When the system verifies the ballot, it sends a confirmation to the voter (step 3). Otherwise, it sends a response indicating the failure of the verification.

6 Implementation

In this section we discuss the implementation details and techniques that can be used for actual implementation of the system. And we weight them in terms of their applicability.

6.1 Criteria

The criteria for assessing a technology for use in the implementation of the project can be summarized in the following points:

6.1.1 Ease of use

Ease of use of the technology involves ease of setting up the development environment, ease of developing and ease of deployment.

6.1.2 Support

Support refers to the size of the development community. Usually indicates the promptness of fixes and the quality of the documentation.

6.1.3 Security

Language traits such as bounds checking, overflow checking, pointer arithmetic limitations contribute to the security of the final product. Some dynamic traits can lead to issues in reasoning about security.

6.1.4 Performance

Due to our use of public key cryptography, CPU and memory efficiency are important.

6.1.5 Portability

Portability is important to satisfy the generality requirement.

6.2 Candidate Technologies for Client Side Code

Client side is most affected by the metrics above. Portability and performance are less of an effect on the server side. Thus, we focus our consideration on the client side.

6.2.1 Web technology (JS/HTML/CSS)

Web technology wins in portability. An internet browser exists on any internet enabled device. JavaScript is popular and well-supported. It has a very large community and very active development. It is the most used language today. JavaScript is also easy to use. Pushing updates to users is also easy for a web application.

JavaScript suffers in terms of security . Any part of the code can be changed at runtime [11] thus enabling the possibility of a part of the code to be overridden silently. It also does not have an SRNG. It has been proposed but is yet to be added to the standard [12]. JavaScript also suffers from performance issues. It has no multi-threading or multi-processing capabilities. It is also an interpreted language which adds run-time overhead.

In conclusion, we believe JavaScript in its current form is not fit for our requirements, though the incredible amount of possible platforms a web service can run on is still very attractive.

6.2.2 Xamarin platform [13]

Mobile applications follow web technology closely in terms of portability and accessibility. The Xamarin platform promises the ability to program for all major platforms using one language (*C#*) and to have common functionality be shared among the platforms. Shared code makes it easier on development time and the auditing of the code.

Xamarin has good support. Ease of use is also not an issue. Performance is also acceptable.

Xamarin suffers from worse portability than web technology. Deployment to users is also worse as a user now needs to install an application. Applications produced by Xamarin also tend to be large in size.

In conclusion, Xamarin is a viable option for the client side, but it is not without its issues. Barrier to entry is high even compared to mobile applications written with a platform's official tools. But it does mean a higher amount of shared code that is written in a managed language and that's where the attractiveness of the platform lies.

7 Conclusion

In conclusion, we have demonstrated the application of a generic e-voting system in a real world scenario where the rules and restrictions have been catered to by extending the concept of holding parallel *yes/no* polls to hold a *1-out-of-L* election to holding parallel *1-out-of-L* elections to have separate sets of candidates. We have designed a system that can provide a great deal of transparency and accountability. The final results of the election can be reproduced by anybody to verify the honesty of the tallier while preserving the anonymity of votes.

We have also discussed the requirements, design and implementation of such a system weighing different technology stacks against each other in specific criteria.

References

[1] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 223–238.

[2] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of paillier's probabilistic public-key system," in *International Workshop on Public Key Cryptography*. Springer, 2001, pp. 119–136.

[3] C. Lambrinoudakis, D. Gritzalis, V. Tsoumas, M. Karyda, and S. Ikonomopoulos, "Secure electronic voting: The current landscape," in *Secure Electronic Voting*. Springer, 2003, pp. 101–122.

[4] I. Damgård and M. Jurik, "A length-flexible threshold cryptosystem with applications," in *Australasian Conference on Information Security and Privacy*. Springer, 2003, pp. 350–364.

[5] K. Peng, R. Aditya, C. Boyd, E. Dawson, and B. Lee, "Multiplicative homomorphic e-voting," in *Proceedings of the 5th International Conference on Cryptology in India*, ser. INDOCRYPT'04. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 61–72.

[6] D. A. Gritzalis, "Principles and requirements for a secure e-voting system," *Comput. Secur.*, vol. 21, no. 6, pp. 539–556, Oct. 2002.

[7] Y.-Y. Chen, J.-K. Jan, and C.-L. Chen, "The design of a secure anonymous internet voting system," *Comput. Secur.*, vol. 23, no. 4, pp. 330–337, Jun. 2004.

[8] M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward a secure voting system," in *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, ser. SP '08. IEEE Computer Society, 2008, pp. 354–368.

[9] D. A. Gritzalis, "Principles and requirements for a secure e-voting system," *Computers & Security*, vol. 21, no. 6, pp. 539–556, 2002.

[10] E. Magkos, M. Burmester, and V. Chrissikopoulos, "Receipt-freeness in large-scale elections without untappable channels," in *Towards The E-Society*. Springer, 2002, pp. 683–693.

[11] T. Ptacek, "Javascript Cryptography Considered Harmful," https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2011/august/javascript-cryptography-considered-harmful/, 2011, [Online; accessed 18-February-2017].

[12] M. Watson, "Web Cryptography API," https://w3c.github.io/webcrypto/Overview.html, 2016, [Online; accessed 18-February-2017].

[13] "Xamarin Homepage," https://www.xamarin.com, 2017, [Online; accessed 18-February-2017].