Mosaic: a secure and practical remote voting system

Takoua Abdellatif*

ESSTHS,
Rue Lamine Abassi 4011 H. Sousse, Tunisia
Fax: (216)73-370-710
E-mail: Takoua.Abdellatif@imag.fr
E-mail: Takoua.Abdellatif@essths.rnu.tn
*Corresponding author

Ahmed Adouani

Proxym-it,
Technopole de Sousse BP 184 Sousse Khezama,
4051, Sousse, Tunisia
Fax: +21673220390
E-mail: Ahmed.Adouani@proxym-it.com

Abstract: This paper presents Mosaic a practical and secure e-voting system. Compared to existing remote e-voting systems, Mosaic is practical since it deals with availability, ease of usage and scalability in addition to the implementation of an efficient security scheme. We demonstrate that the adaptable architecture of Mosaic system enforces the system scalability and performability by evaluating the system on the French grid, grid 5K. An embedded management system allows for Mosaic self-adaptation in front of failures or security attacks without breaking the security properties of the system.

Keywords: e-voting; component-based distributed systems; secure autonomic system.

Reference to this paper should be made as follows: Abdellatif, T. and Adouani, A. (2014) 'Mosaic: a secure and practical remote voting system', *Int. J. Autonomic Computing*, Vol. 2, No. 1, pp.1–20.

Biographical notes: Takoua Abdellatif received her Engineering degree from the ENSIMAG School (INPG) in Grenoble in France. She received her PhD in 2006 from INPG and INRIA Grenoble in France. She worked for five years in Hewlett-Packard Company as a Research and Development Engineer. She is currently an Associate Professor at the University of Sousse, Tunisia. Her research interests are focused on building autonomic secure distributed systems. She is currently working in collaboration with DCS/Verimag team in Grenoble.

Ahmed Adouani is a Research Engineer in ProxymIT which is an international company working on new technologies. He received his Engineering degree in 2008 from the University of Sousse and he is currently leading the mobile software development team in ProxymIT.

1 Introduction

1.1 Problem statement

E-voting system design and implementation is currently an active research subject. Indeed, e-voting systems are complex distributed systems that have to fulfil challenging security properties: universal verifiability, coercion-resistance and voter privacy. We intend by universal verifiability the ability to audit the voting process at any step. The coercion-resistance means that the voter cannot be influenced by any attacker at voting time. Finally, the privacy property hides any mapping between the voter and his choice. The challenge in building e-voting protocols consists in ensuring simultaneously these three properties. Furthermore, ensuring coercion-resistance is difficult especially for a remote voting context where the voters use their browsers to vote instead of voting in a close and protected classical voting room. Juels, Catalano, and Jakobsson (JCJ) (2005) proposed the first scheme that considers real-world threats and that is more realistic for remote elections. The scheme mitigates coercive attacks by allowing the voter to deceive the adversary about her true vote intention and to vote again afterwards. Nevertheless, it has a quadratic work factor (in number of votes) to compute the voting results and thereby is not efficient for large scale elections. The evaluation result of an improved version of this security protocol implementation (Clarkson and Chong, 2008) confirms this limitation. Based on the work of JCJ, Araujo, Foulle and Traoré (AFT) (2007) recently proposed a coercion-resistant protocol that has linear work factor. The question is whether this linearity is confirmed by the protocol implementation and whether it is sufficient for system scalability. Furthermore, a set of implicit security assumptions are considered in the protocol. In the implementation, they need to be explicitly defined and their implementation limitation identified. Finally, other practical aspects need to be addressed when designing the system, mainly the system management that is the configuration, deployment and the system adaptation to external events such as failures or security attacks. The system management needs to be itself secure and has to respect the security properties of the e-voting system.

1.2 Contribution

This paper presents the design and the evaluation of Mosaic, an adaptable e-voting system that implements the AFT security protocol. Mosaic uses a component-based architecture that helps the system to provide the following practical features:

- *An embedded secure management system*: The system installation of an e-voting system requires the setting of a large number of cryptographic and configuration parameters on distributed machines. Indeed, the implemented protocol supposes that some cryptographic parameters are set before starting the system. If not automated, this task can be tedious especially with a large scale system with hundreds of machines. As a component-based system, the system configuration and deployment in Mosaic are automated thanks to the use of an architecture description language (ADL) that allows setting configuration parameter values. In addition to the system configuration and deployment, the management system adapts the voting functional system to external events such as security attacks or system failures. As explained in

Section 4, the management system communications are secure preserving the overall system security properties.

- *System availability*: Mosaic components that present unresponsiveness or security failures can be detected and isolated without stopping the whole system. Nevertheless, this supposes that only a threshold number of components are allowed to fail at the same time for two reasons. First, some decryption operations rely on distributed shared keys between a set of components; a threshold of them needs to be available for these encryption operations. Second, the failure of a component leads to the system security degradation. Therefore, only a limited number of components are allowed to fail simultaneously. Otherwise, the system needs to be restarted.
- *Usability*: E-voting system usability is as important for some voters as the system security (Herrnson et al., 2008). Indeed, if the system requires the usage of complex cryptography data, the system can be rejected by users even it provides all security guarantees. Indeed, the e-voting system has to be easy to use to encourage common people to vote. The security protocol that Mosaic implements, provides to voters convenient ways to cast their vote since the voter needs only to retain a password to cast his vote. We demonstrate how this feature is maintained in Mosaic implementation. The usability issue does not only consider voters but also the system administrators. Thanks to the component-based architecture of the system and of the management system, a human administrator can easily set up the system configuration and the management policies using the same tools. The administrator can also supervise all the system execution steps through the component control interfaces.
- *Scalability*: A performance evaluation on the French grid, grid 5K ('The grid 5K', 2009), demonstrates that the theoretical linearity of the protocol can be obtained in a real large scale distributed environment only if the cryptographic libraries and database access are efficiently implemented. Furthermore, we demonstrate how Mosaic design allowing for task processing parallelisation reduces the processing time. We also measure the management system overhead and the time we gain thanks to the fault-tolerance feature of the system. In addition to the performance evaluation in the nominal sate, we demonstrate that failures do not lead to visible performance degradation which ensures the system *performability*.

1.3 Related work

Many researchers focus on defining security protocols and cryptographic solutions e.g., (Chaum, 1981; Park et al., 1994; Furukawa and Sako, 2001; Boneh and Golle, 2002; Jakobsson et al., 2002; Golle et al., 2002) for e-voting systems, but few implementations focus on handling practical issues like usability, administration, fault tolerance and scalability. The system that is the more comparable to ours is Civitas (Clarkson and Chong, 2008) since it provides the same security e-voting properties (universal verifiability, coercion-resistance and voter privacy). Like ours, Civitas security scheme is based on JCJ scheme. Civitas is a first implementation experience for a JCJ-like scheme showing the implementation feasibility of such security protocol. In Mosaic, we address more advanced properties such as scalability, availability and usability. The scalability in Civitas relies on splitting the voters to a number of blocks; each block represents a

geographical region but does not address the way this can be done. The large-scale deployment of e-voting systems requires a secure and efficient management system automating the configuration, deployment and runtime adaptation. To our knowledge, Mosaic is the first e-voting system that proposes such a management system. Furthermore, within a geographical region, the number of voters can be important (more than 10,000). For this reason, other solutions are necessary for scalability within a region.

Helios (Adida, 2008) is a web-based voting system that provides universal verifiability and voter privacy but not coercion-resistance which simplifies the system underlying protocol and design. REVS (Lebre et al., 2004) implements a protocol resisting to malicious servers. Nevertheless, it does not handle the system failures and scalability. In Lundin (2008), the author motivates the adoption of a component-based architecture for implementing modular e-voting systems. The system modularity gets the development and the maintenance of each system part easier; each part can independently be designed, coded and verified. Nevertheless, in Lundin (2008), the system modularity is not used for runtime adaptation and scalability like in Mosaic.

Our work is also related to adaptable and component-based systems (Subramonian et al., 2007; Cheng et al., 2004; Abdellatif et al., 2007; Bouchenak et al., 2005). Like in Mosaic, these systems use software components as units of configuration, deployment and reconfiguration for the management and the managed systems. In our work, we additionally address the security management issue which requires securing the communication between the different system components. Indeed, since Mosaic is a security system, its management and adaptation have to maintain the system security properties. We believe that the security mechanisms and protocols implemented in Mosaic can be reused to secure the management of other component-based distributed systems.

The rest of the paper is structured as follows. Section 2 describes the security protocol that Mosaic implements by default. In Section 3, we present Mosaic design and in Section 4, the security model and assumptions. In Section 5, we describe Mosaic management system and its adaptation mechanisms and policies. In Section 6, we present the system usability feature. Section 7 illustrates the experimental environment and performance evaluation results and we conclude in Section 8.

2 Security protocol

Unlike many voting protocols e.g., (Chaum, 1981; Park et al., 1994; Furukawa and Sako, 2001; Boneh and Golle, 2002; Jakobsson et al., 2002; Golle et al., 2002) requiring user authentication and based on the same idea as JCJ (Juels et al., 2005), AFT (Araujo et al., 2007) scheme proposes an indirect authentication and authorisation mechanism; user authentication may be exploited by coercers to influence the voter. Therefore, a voter does not authenticate directly at the time he casts his ballot, e.g., by attaching a signature. Instead, he casts his vote together with a non-deterministic *encrypted credential*, i.e., an encrypted secret value known only to the voter and the election authorities. This credential has a mathematical structure ensuring that even if an adversary possesses many valid credentials, he cannot obtain a new valid one. This way, a voter over coercion can make a fake credential to deceive an adversary who cannot distinguish between a fake and a valid credential.

The scheme involves four kinds of servers that execute their tasks in four steps.

2.1 Voting system servers

Mix servers

Mixnets were introduced by Chaum (1981) in the early 1980s. It was initially proposed as a mechanism to prevent the association between the senders and the receivers in electronic mailing system. Mixnet typically consists of a series of mix servers and relies on public key cryptosystem such as ElGamal or RSA. Each mix server performs by decrypting (or re-encrypting) its inputs and by reordering the inputs randomly and secretly. With these techniques it is difficult for an eavesdropper to determine which output messages correspond to which input messages. Therefore, the choice of the sender is kept anonymous. Thanks to the encryption of the votes, the mix server cannot know the mapping between voters and their votes. In electronic voting system, the Mixnet is desired to be verifiable, which means that the correctness of work by each mix server can be verified. For this reason, each mix server provides the proof that the shuffling work was done without modifying the input list. Note that the security privacy is enforced when the number of mix servers is important.

Talliers

They are responsible for verifying the Mixnet proofs and for computing the voting results. They collaborate to perform decryption information by sharing private keys. A threshold number of these talliers are necessary for performing a decryption task. This means that only a threshold number of talliers need to be available and non-suspicious.

Registrars

These authorities have the task of issuing a valid credential to each eligible voter. Also, they help the talliers identifying valid credentials.

Bulletin board

For verifiability, the calculated data and proofs are stored in the bulletin boards which are servers managing and publishing the voting data. Once receiving information, the bulletin board stores it and cannot delete or modify it.

2.2 The scheme phases

Setup phase

In this phase, the general voting parameters are established and published along with a digital signature on the bulletin board as well as the list of the voting candidates.

Registration phase

After verifying the voter is eligible, the registrars issue to the voter a secret credential via an untappable channel where the possible coercer is considered absent.

Voting phase

The voter casts her ballot by sending a tuple containing the credential as well as a set of proofs which ensure that the vote is well-formed.

Note that the Registrars and talliers rely on a modified version of El Gamal Threshold cryptosystem defined in Juels et al. (2005). In the threshold version, the El Gamal public key and its corresponding private key are cooperatively generated by n parties (in our case, the talliers and Registrars); though, the private key is secretly shared among the parties. In order to decrypt a ciphertext, a minimal number of t out n parties is necessary.

Tallying phase

In order to compute the voting results, the talliers perform the following steps:

1 Verifying well-formedness proofs: The talliers verify the proofs on each tuple and remove the invalid tuples corresponding to invalid candidates.

2 Removing duplicates: The valid ballots are stored into a public hash table. Each ballot is considered as the value and a particular field in the ballot is considered as the key. If a collision is found, a pair of duplicate ballots is detected and only the recent one is kept.

3 Mixing preparation: The ballot fields used to eliminate duplicates are eliminated from each ballot. The talliers then prepare the ballots to the mixing phase by collaboratively encrypting all the fields.

4 Mixing tuples: The last tallier participating to the encryption publishes the encrypted list on the bulletin board and sends it to the first mix server in the Mixnet. Each mix server mixes the ballot list, re-encrypts the ballot fields and calculates the proof that the mixing does not modify the ballot list. The mixed and re-encrypted list as well as the proof are then stored in the bulletin board. The protocol proposes Neff (2004) Mixnet as an example of universally verifiable Mixnet which is implemented in Mosaic.

5 Identifying valid votes: For each tuple, the registrars and talliers collaborate to check the validity of the ballot: The registrars calculate an encrypted number using the secret credential of the voter and the talliers decrypt collaboratively this number (without revealing the value of the credential). If the decryption is equal to one then the credential is a valid adversary.

6 Decrypting and counting the votes: The talliers employ their secret shared key to cooperatively decrypt each field of each tuple with valid credential. After that, one of the talliers counts the votes and publishes the results on BB. In the protocol implementation, all these steps are classified into fine grained tasks; each task is identified with a unique identifier.

3 System design

As illustrated in Figure 1, Mosaic is composed of two separate layers: a management layer represented with the *administrator* component and the functional layer containing

Mosaic servers. The two layers are designed as *composites* containing *subcomponents* constructed using the same component model, Fractal (Bruneton et al., 2004). Components let us build modular system architectures with explicit dependencies between system parts. Each component provides two types of interfaces. Functional interfaces bind components with each other and with possibly remote bindings between components, and they invoke the components' functional methods. Control interfaces manipulate the configuration properties, life cycle, and components' other non-functional aspects. Through component abstraction, we can explicitly define the architectures of the system to deploy. To express the desired architecture, we use an ADL that describes the system components in a uniform way and describes their relationship in terms of bindings and encapsulation. Fractal is a general component model that distinguishes two types of components: primitive and composite. Primitive components are standard Java classes that conform to certain coding conventions. Composite components encapsulate a group of primitive or composite components. The system architecture, written in the fractal ADL, is expressed in terms of the component model bindings between components and containment relationships. Bindings between components can be local (in a single Java virtual machine) or remote. Furthermore, bindings can be dynamically established and broken which is practical for building runtime adaptable systems. These properties are specific to fractal, the reason why we chose it to build our system.

Figure 1 Mosaic architecture (see online version for colours)

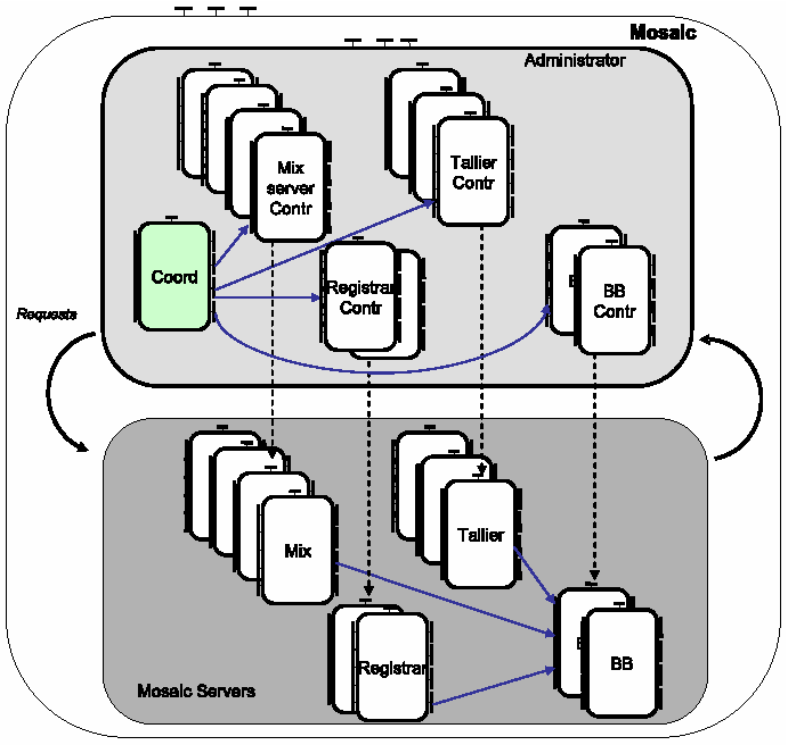
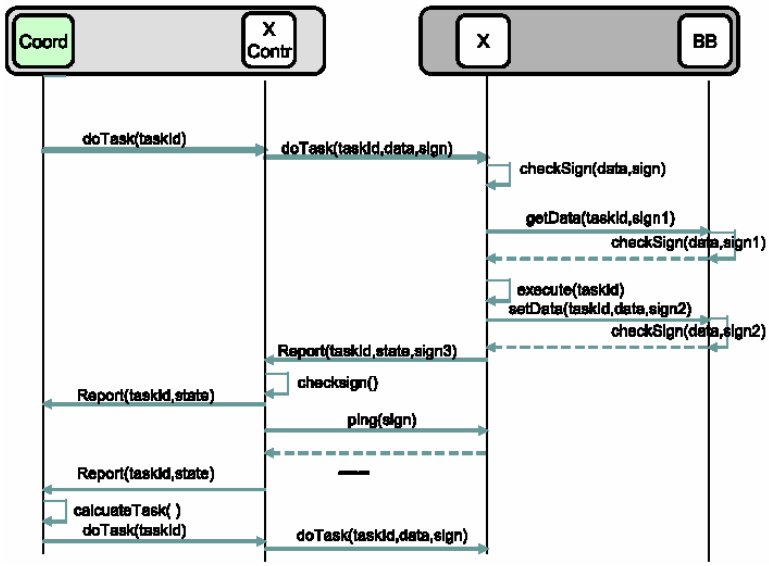


Figure 2 Communication bindings and interfaces (see online version for colours)



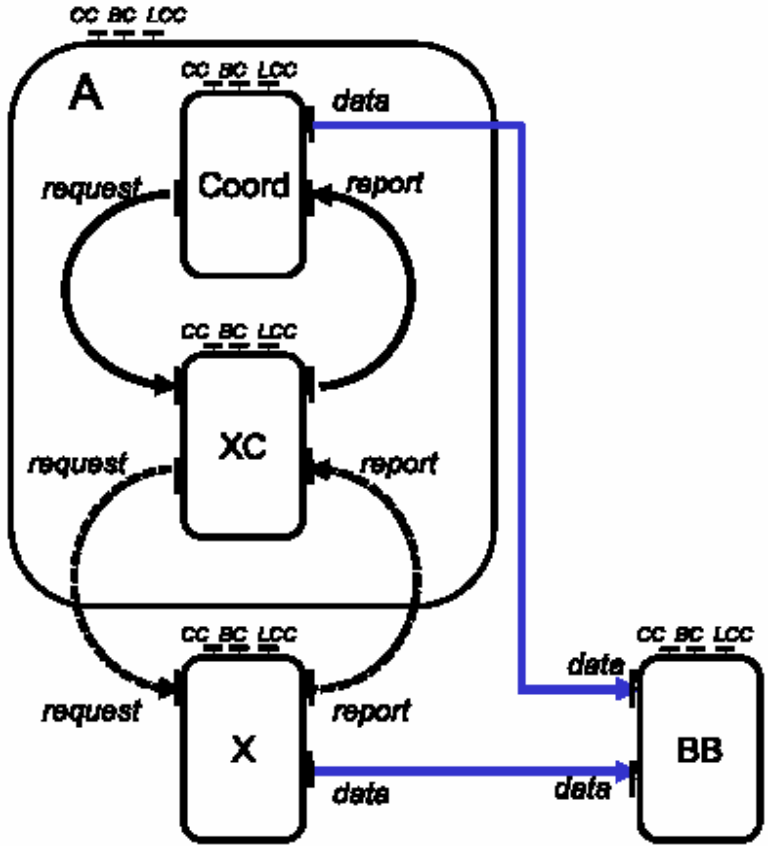
Mosaic is an assembly of two kinds of components: the management components contained in the administrator composite and the functional components encapsulating the voting servers: the registrars, talliers, mix servers and the bulletin board components. The first category of components (the controllers and the coordinator) is in charge of controlling the system at runtime. The servers represent the ones described in the security protocol. Encapsulating each server in a component means that its configuration, that becomes uniform, uses the component control interfaces and the communication with other parts of the system use the component bindings through the communication interfaces.

The administrator is a composite comprising the controllers of the server components. In Figure 1, the controllers are denoted *X Contr* where X can be a mix server, a registrar, a tallier or a bulletin board. Each controller component supervisors its associated server. Mosaic servers components implement the security scheme as a set of tasks; each task is identified by a unique identifier. The coordinator component, denoted *Coord*, orchestrates Mosaic tasks and implements the adaptation policies. Figure 3 presents the communication flow between the management and managed layer. To execute a task, the coordinator sends a task request containing the taskID and optionally some other data to the appropriate coordinator, XC, that forwards the task to the component server X. When a functional component executes its task, it sends back to its controller a report giving indications about the task execution state (done, failed or pending). The server component X gets data from the BB and puts the calculated result in the BB afterwards. For instance, when a mix server is asked to make a shuffle of the voters list, the correspondent controller forwards the coordinator request to the mix server component that gets the list from the BB, makes the shuffle, calculates the proofs, puts the result list on the BB and sends a state report telling that the task was correctly performed. Similarly, to ask a tallier to verify a proof calculated by a mix server, the coordinator sends a request to the tallier controller that transfers the request to the tallier component.

The later gets the proof data from the BB, verifies the proofs and sends back to the coordinator the verification result. The communication security will be explained in 4.3.

In addition to request forwarding, the controller component role is to check the availability of the servers by regularly pinging them. If no response is detected within a configurable time, the controller informs the coordinator about the non-responding server. The coordinator performs the appropriate adaptation operation described in Section 5.

Figure 3 Communication bindings and interfaces (see online version for colours)



4 System security

Mosaic implements the security requirements defined in the protocol. Indeed, the Universal verifiability is ensured by publishing the data and the associated proofs of each step on the BB to allow anyone to verify them. Privacy of voters is provided by the mix servers. The coercion-resistance is ensured thanks to credentials usage allowing to the voters to deceive the coercers and to the system to distinguish valid credentials from invalid one. In our work, we suppose that this protocol is correct.

Nevertheless, the protocol supposes a set of assumptions. When designing a security system, all these assumptions need to be explicitly presented and the implementation limitation identified. We classify these assumptions into the following points: the adversary behaviour during the voting time, the cryptographic key management and the communication security between components.

4.1 Assumptions on the coercer behaviour

The protocol supposes that the adversary may force the voter to vote for him. It also supposes that the registration phase is free of adversaries and is performed using an untappable channel. In practice, we suppose that the voter goes physically to register and provides his secret password, necessary to the Registrars to generate credentials. We suppose that this step occurs without the presence of the adversary. According to the protocol, the credential, necessary for voting, is composed of two parts: the secret password easy to retain and a public part of the credential that is longer and that can be provided on a CD or sent to the voter by e-mail.

4.2 Cryptographic key management

Before starting the system, the servers need to have a set of cryptographic data installed, mainly the public and secret keys as well as a set of secret credentials. The distribution of such secret data in a distributed system is a real issue since it has to ensure the confidentiality of secret information and the integrity of public one. These data are necessary for encrypting, decrypting, tagging or signing exchanged messages. We suppose that all this data is installed on the system machines before starting the system. Their security can be accomplished using existing security tools such as the trusted platform module (TPM) (2009). The TPM is a hardware-based security and cryptography chip that can store such cryptographic data on the servers in a confidential way. Regarding the secret in-memory data such as code and configuration parameters, we make the assumption that the probability to corrupt a part of a machine memory is very weak.

4.3 Component communication security

The security scheme assumes that the communication network is secure. In this section, we explain how this assumption is implemented in Mosaic. Figure 3 presents the main communication interfaces and bindings between Mosaic components. In the figure, X component represents either a mix server, a tallier or registrar component. A component has two kinds of interfaces: control interfaces and communication interfaces. The control interfaces are: the configuration controller to set configuration parameters, the binding controller (BC) to setup or break the bindings with other components and the life cycle controller (LCC) to manage the life cycle of a component. All functional components are linked to the administrator through the request and report interfaces and linked to the BB through the data interfaces to get or set data.

The communication bindings have to ensure the integrity of the exchanged messages. We intend by integrity the following:

1 the sent messages are not modified in the network

2 the actual sender is one of the expected components (and not another program attacker).

To achieve these integrity aspects, all messages are signed by the component sender. The receiver component has the list of expected senders configured at the bootstrap time and their public keys; it can therefore identify the sender. The message signature cannot avoid the replay attack; an attacker can reuse an exchanged message between two components. Replay attacks can lead to the system denial of service. For example, if a mix server receives a set of replayed requests to shuffle messages, it can not be available for valid requests from the administrator component.

To avoid replay attacks, we tag the messages with nonces (Anderson, 2001). For each task, the administrator generates a dedicated nonce as a concatenation of a random number and a sequential number. The table associating taskID and their nonce is stored at the bootstrap time in the BB. The nonces are public; since messages are signed, any attacker message will be rejected even it contains a valid nonce. Each component X checks its task nonce from the BB. Each task involves four exchanged messages: the administrator component sends a request to X, X gets the task data from the BB, X sends the result data into the BB and sends the report to A. These messages use the same nonce that expires when the four messages are accomplished. If one of the messages is replayed, the component receiving the message can detect that the nonce expired (since already used) and reject the message.

If for any reason (mainly failure recovery), a task needs to be executed again, the administrator creates a new nonce for the task and stores it in the BB. Thanks to the sequential number in the nonce, a component can detect that a new nonce is generated for the task and checks it from the BB. If a message with an old nonce is replayed, the sequential number shows that the nonce has expired. Note that the communication between the administrator and the BB use a set of predefined nonces installed at the bootstrap time on the administrator and the BB components.

For universal verifiability, all data has to be publically published on the BB, the reason why we store the nonces in the BB. This security protocol relies on two assumptions: the administrator component and the BB are not corrupted. It is also important that the administrator component and BB are available.

This is ensured in Mosaic by the replication of these components and the active replication of their state as we explain it in Section 5.2.

5 System management

5.1 Configuration and deployment

As a component-based system, Mosaic configuration and deployment are based on the ADL. In the system ADL file, the number of mix servers, registrars and talliers components depend on the desired security level that is proportional to the number of these components. In this file, the bindings are established following Figure 3: The controllers are connected to their associated servers that are connected to the BB. For each component, an ADL configuration file defines the following cryptographic parameters:

1 the reference to the component private key (e.g., a reference to the TPM storing the key)

2 the list of senders that is the components whose messages are accepted and the reference to the public key of each sender.

Additionally, the coordinator component configuration allows defining the desired mixing mode: optimistic, pessimistic or relaxed. In the optimistic mode, the shuffling and verification tasks are parallelised; the ballot list shuffled by a mix server is directly sent to the next mix server and the shuffling verification is done in parallel by the talliers. In Section 7, we demonstrate that the optimistic mode improves significantly the system performance. In the pessimistic mode, the shuffled list is sent to the next mix server only if the task proof is verified and checked valid. Finally, in the relaxed mode, the proof validation is not required for all mix servers; some mix servers, considered trustful (e.g., if they belong to the administrator domain) are not required to provide the proofs of their shuffle. Section 7 demonstrates that task verification is costly and adopting a relaxed mode, when possible, can improve the system performance. Furthermore, it is possible to configure the number of servers allowed to simultaneously fail. For the talliers and registrars, a threshold number of servers need to be available. For the Mixnet, the number of available servers depends on the desired security level.

For the system deployment, a bootstrap program is needed on the target machines to install the necessary code and instantiate the components. We also suppose that the cryptographic keys are available following the addresses mentioned in the configuration references, otherwise the system deployment fails.

5.2 System adaptation

In Mosaic, we apply the adaptation mechanisms when one of these events occurs:

1 when a server does not respond to its controller pinging during a timeout; it is therefore considered failed

2 when a server does not provide correct proofs of its task execution; it is then considered suspicious.

For the administrator and BB components, considered safe, only fail-stop faults are considered. The failure recovery protocol is implemented in the coordinator that orchestrates Mosaic processing following a well-defined order of tasks. When a component-failure occurs, the coordinator decides how to reorganise the tasks accordingly. In this section, we present the adaptation policies and mechanisms following the kind of failed components.

5.2.1 Mix server failures

When a mix server component is detected failed, its controller notifies the coordinator. The current adaptation policy consists in *isolating* the faulty component that becomes no longer involved in the voting process. Another policy could be to replace the failed component with a new one. Nevertheless, this requires handling carefully the trust between the introduced component and the rest of the system as well as the component cryptographic key exchange. Key management is not currently introduced in Mosaic as

said earlier in Section 4, the reason why we rather use the isolating policy. The number of mix servers allowed to fail is determined by configuration. If one more mix server, the hole system, considered unsafe, is restarted. Otherwise, a faulty mix server, not involved in a task is removed from the system. In that case, the coordinator unbinds the mix server from the BB and the mix server from its controller. Breaking these bindings leads to the sender tables update at the administrator and BB components. The mix server component is removed from the administrator and BB senders list and its future messages will be automatically rejected.

If the mix server fails when involved in a shuffling task or when it fails before providing the proofs of a previous shuffled task, the coordinator applies a *rollback*; the system is returned to the state just before failed mix server shuffling task. Indeed, in the case of an optimistic mixing, that is the mixing parallelising the shuffling and the proof verification, a non-verified shuffling task T provides a non-valid list used in the tasks following T. Therefore, the cancellation of T involves cancelling all the tasks done after T. For that, the coordinator stops the running component, isolates the faulty mix server and all the lists and proofs stored from T are positioned invalid. The coordinator resumes the system processing beginning from the component following the failed server. In the case of optimistic mode, this can involve replaying the execution of some tasks. In this case, the administrator generates new nonces for the tasks to redo and store them on the BB. Note that all the control messages (stopping, unbinding) are performed by the administrator component with the ongoing task nonce. They are therefore unique and cannot be replayed by an attacker.

5.2.2 The registrar and tallier failures

The registrars as well as the talliers collaborate to perform a proof verification or calculation. The security scheme requires that a threshold number is available; the coordinator restarts the system otherwise. If one of the servers fails when running a task, this task is ignored and the server is isolated.

5.2.3 The administrator failure

For fault tolerance, the administrator component, let us call it A, is actually replicated. All the task states are actively replicated on a replica component B. B sends regularly heart beat messages to check A availability. When a failure occurs, B coordinator subcomponent resumes the system processing from the last validated task. To achieve this goal, B breaks the bindings between A and the server components and establishes new ones between them and B. Note that, at bootstrap time, the functional components has B in the senders list. Therefore, they accept the requests from B and send their reports to it. To send a request, B uses the nonce available at the BB and that are recognised by the rest of system components.

5.2.4 The bulletin board failure

Like for the administrator component, the BB component is replicated for fault tolerance. The BB failure can be detected by the BB controller that notifies the coordinator. The administrator stops the running component, unbinds the functional components from the failed BB using the BC of Mosaic composite. It starts the BB backup and establishes the

bindings with the new BB. The data base availability is out of our work scope. It can be ensured with classical active replication and clustering techniques.

6 Mosaic usability

For a remote voting system, it is important that the system is usable during the voting time by voters and during the tallying time by auditors and also the administrators.

Regarding voters, thanks to the security protocol, voting with Mosaic is very convenient. Indeed, like in e-business Web applications, Mosaic provides an applet allowing the following tasks:

1 The voter chooses his favourite candidate and validates his choice.

2 He introduces his NIC number and validates his choice.

3 He selects the file with cryptographic parameters he obtained on a CD or by e-mail at registration phase.

4 He can enter at this step, his secret phrase. If a coercer person is present, the voter can enter a fake phrase that will be accepted by the system. The voter can revote at any time he wants, and the last valid vote (i.e., with a valid secret phrase) will be taken into account at the final tally.

5 The voter will be invited to confirm the previous steps and click on 'CAST MY VOTE' button. Before submitting his vote, the voter can return to any step to verify or correct something.

For universal verifiability, all the public data on the BB are available through a graphical interface. Each task data and proofs are stored on the BB. The auditor can therefore, check all the proofs, even the rejected ones.

The modular architecture of Mosaic and its self-adaptation at run time simplify considerably the role of the human administrator. Indeed, his role consists mainly in configuring the system through the writing of the ADL files. The deployment is automated thanks to the component deployment tool. At runtime, failures are automatically detected and repaired. Nevertheless, the human administrator can interact with the system at run time through the control interfaces using the graphical interface that is automatically generated from fractal component-based system. He can decide to stop components, establish bindings or break ones. For example, mix servers number can be reduced, even if the system already started, if the administrator needs to do so for a compromise between performance and security.

7 Performance evaluation

In the experimental evaluation, we measure the processing time for a number of votes varying from ten to 1,000 votes for the three phases:

1 the preparation phase comprising the elimination of replica and non-well formed ballots

2 the mixing phase

3 the final tally phase comprising the validation of ballots, the decryption and counting.

We also evaluate the time gained thanks to the optimistic mode configuration for the mixing phase and evaluate the tallier and registrar number impact on the system scalability. Furthermore, we measure the management system overhead. Finally, we evaluate the system performability by measuring the time spent to recover from a failure.

7.1 Experimental environment

Our experimental platform consists of 79 machines of grid 5K (‘The grid 5K’2009) (Lyon cluster). The hardware configuration of the machines is as follows:

- Hardware: Sun Fire V20z
- CPU: AMD Opteron 250 2.4GHz / 1MB / 400MHz
- 79 nodes × 2 cpus per node = 140 cpus
- Memory: 2 GB
- Network: Gigabit Ethernet
- Software configuration: openjdk-6, mysql 5.0.75-1 server.

99% percentage of Mosaic code is written in Java and the calculation part (computation of BigInteger, exponential calculation) is native and wrapped in Java using JNI. For the native library, we used the version 4.1.4 of GMP. We used the following option ‘java – server – Xms800m’ to launch the JVM for Mosaic. We consider the following configuration: four machines for the mix servers, four machines for the talliers, one machine for the BB, one machine for the registrar and one machine for the administrator.

7.2 Experimental results

Figure 4 confirms the linearity of Mosaic scheme results. These results are actually obtained thanks to two optimisations:

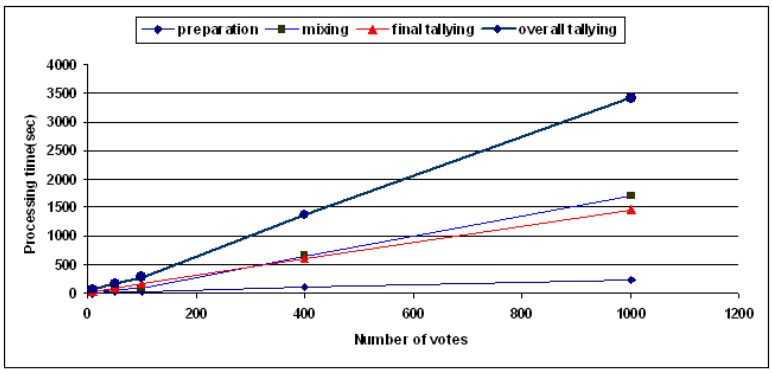
1 The usage of a native library for the BigInteger calculation

2 the optimised access to the database.

Indeed, the first results with 100% Java code show a Mosaic execution time seven times more slow than the optimised version. Furthermore, we identified that the data base represented the bottle neck when the number of votes exceeds 100 votes. We solved the problem by splitting the data to store in the data base into different chunks. It is necessary to find the appropriate chunk size to optimise the number of messages. In the results, we present in this paper, we choose arbitrary 150 Kbytes. We believe that these results can be significally improved if a better access mechanism to the database (like caching or clustering) is introduced. Furthermore, more efficient machines can also improve the calculation time and therefore the overall performance of the system. Using our experimental environment, the overall tallying time does not exceed 25 mn for 400 votes and one hour for 1,000 votes. To our knowledge, this is the best performance results of

existing e-voting systems with equivalent security properties and comparable hardware configuration.

Figure 4 Performance evaluation of Mosaic with a configuration of four talliers, four mix servers, one bulletin board and one administrator (see online version for colours)

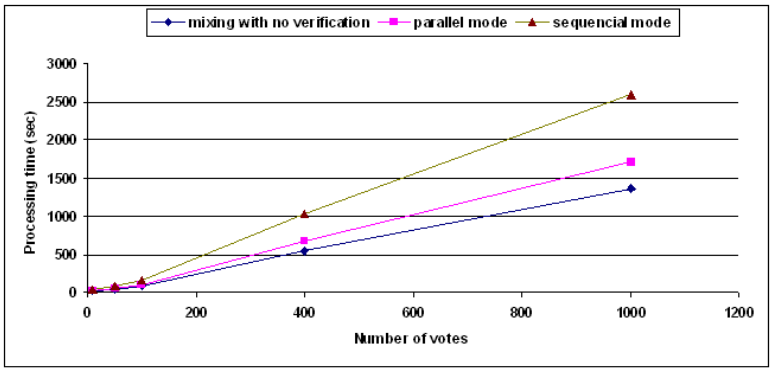


Mixing mode comparison

In Figure 4, the mixing phase is implemented in the optimistic mode where the shuffling and verification tasks are parallelised. Figure 5 shows that the optimistic mode allows gaining more than 50% of the mixing time for 1,000 votes. Furthermore, an evaluation of mixing without verification demonstrates that verification requires 50% of the global mixing time. When possible, relaxing the security by not verifying all the proof verification can improve the overall system performance.

Note that another way to optimise the system performance is to split the votes list into different parts and distribute each part of the task to a free tallier. Examples of tasks are the verification of the well-formedness of the ballots and the combination of the decrypted parts. For the tallier parallelisation, a precise scheduling algorithm is needed. In our evaluation, we only focused on the evaluation of the parallelisation between the mixing and its verification tasks.

Figure 5 Mixing phase optimisation (see online version for colours)

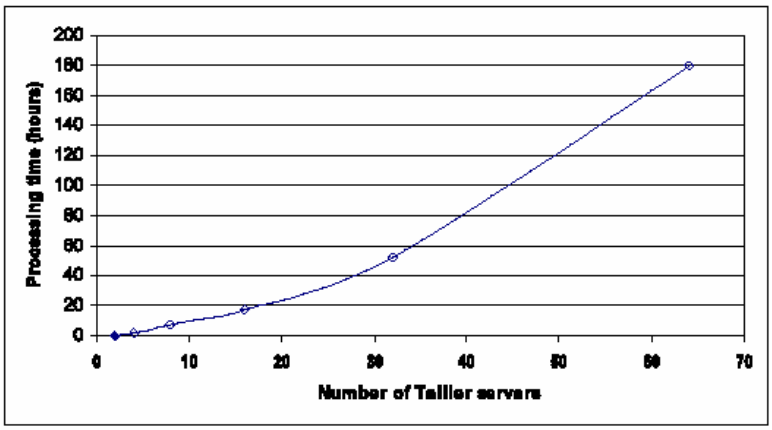


Server number impact on performance

Figure 6 represents in hours the processing time when varying the number of mix servers. The number of the talliers is kept constant and equal to four as well as the number of votes is fixed to 1,000. As expected, the processing time increases in a linear way as the number of mix servers increases. Figure 7 represents the execution time as a function of the number of talliers. The tallier task execution is more complex than the mix servers, the reason why the processing time is rather exponential when the number of talliers increases. The number of mix server equals four servers and the number of voters is fixed to 1,000. This study shows that the number of servers can have a considerable impact on the processing scalability. A compromise about the desired security level and the system performance has to be considered. Note that in the previous experiments, the number of four for each kind of the servers is the common configuration in current e-voting systems and we demonstrate that with the same configuration, we obtain significant better results.

Figure 6 Mix server number impact on the system performance (see online version for colours)

Figure 7 Tallier server number impact on the system performance (see online version for colours)



Management overhead

The system management time is spent in the signature preparation, message sending and the signature verification for each exchange. A signature consists of two operations of comparison, two multiplications, and two modular exponentiation. A signature verification requires the same time than the signature preparation. This time is constant of few milliseconds but globally insignificant compared to the other kind of tasks (proof preparation, proof verification, a mixing operation, an encryption or decryption). This result shows that the management layer has no overhead on the overall system performance.

Performability

As explained in Section 5.2, the system is self-healing. A 'roll back' policy consists in cancelling a set of tasks when a failure occurs or a suspicious component is detected. Evaluating the system performability consists in measuring this 'roll back' cost that is the cancelled task time. In the sequential mixing mode, the failure of a mix server which leads to loosing the time necessary for a mixing task and a proof preparation. In the worst case, this costs 3.5 mn. In the optimistic mode, the 'roll back' cost is about 7 mn. The tallier and the registry failure costs 2 mn for each component. Regarding the BB or the administrator failures, restarting the server backup and updating the address tables in the other servers require approximatively few seconds. Therefore, the failure of a component does not have a significant impact on the overall execution time.

8 Conclusions and future works

This paper shows that adopting an autonomic architecture in building e-voting systems is an efficient solution to enforce the scalability properties defined in theoretical e-voting security schemes. Indeed, the modular component-based architecture of Mosaic allows a fine-grained control of each task by controlling separately each component. This allows for parallelising a set of task which has an immediate gain on the system performance and scalability. In this work, we focus on parallelising the tasks of the mix servers. Nevertheless, many other tasks can be easily parallelised thanks to Mosaic modularity. Furthermore, the control at the granularity of the component, allows for the system self-healing with a low cost. This ensures the performability of the system, the property that distinguishes Mosaic from the existing e-voting systems. Existing systems generally restart the whole system when failures occur which is costly when many failures occur at execution time. In the current version, the adopted recovery policy consists in isolating the faulty component. This may lead in some cases to the security level degradation, e.g., when decreasing the number of mix servers. Another policy could consist in replacing the faulty component with a new one. This requires a secure management of key distribution and management which is one of our future works. Furthermore, as a consequence of the component-based architecture of Mosaic, the system configuration and deployment are automated using the ADL files. Finally, as a future work, we plan at relaxing some assumptions considered in the current version like the safety of the administrator and bulletin board components.

References

'The grid 5K' (2009) March, available at https://www.grid5000.fr.

'The trusted platform module' (2009) March, available at https://www.trustedcomputinggroup.org/.

Abdellatif, T., Kornas, J. and Stefani, J.B. (2007) 'Reengineering j2ee servers for automated management in distributed environments', *IEEE Distributed Systems Online*, Vol. 8, No. 11.

Adida, B. (2008) 'Helios: web-based open-audit voting', in *Proceedings of the Seventeenth Usenix Security Symposium (USENIX Security 2008)*.

Anderson, R. (2001) *Security Engineering: A Guide to Building Dependable Distributed Systems*, Wiley, ISBN 0471389226.

Araujo, R., Foulle, S. and Traoré, J. (2007) 'A practical and secure coercion-resistant scheme for remote elections', in *Frontiers of Electronic Voting*.

Boneh, D. and Golle, P. (2002) 'Almost entirely correct mixing with applications to voting', in *CCS '02: Proceedings of the 9th ACM conference on Computer and Communications Security*, pp.68–77, ACM, New York, NY, USA.

Bouchenak, S., Boyer, F., Hagimont, D., Krakowiak, S., Mos, A., de Palma, N., Quéma, V. and Stefani, J-B. (2005) 'Architecture-based autonomous repair management: an application to J2EE clusters', in *24th IEEE Symposium on Reliable Distributed Systems (SRDS)*, October, Orlando, Florida, USA.

Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V. and Stefani, J-B. (2004) 'An open component model and its support in Java', in *Proceedings of the International Symposium on Component-Based Software Engineering (CBSE'2004)*, Edinburgh, Scotland.

Chaum, D.L. (1981) 'Untraceable electronic mail, return addresses, and digital pseudonyms', *Commun. ACM*, Vol. 24, No. 2, pp.84–90.

Cheng, S-W., Huang, A-C., Garlan, D., Schmerl, B.R. and Steenkiste, P. (2004) 'Rainbow: architecture-based self-adaptation with reusable infrastructure', in *1st International Conference on Autonomic Computing (ICAC 2004)*.

Clarkson, A.C.M.M.R. and Chong, S. (2008) 'Civitas: toward a secure voting system', in *IEEE Symposium on Security and Privacy*.

Furukawa, J. and Sako, K. (2001) 'An efficient scheme for proving a shuffle', in *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pp.368–387, Springer-Verlag, London, UK.

Golle, P., Zhong, S., Boneh, D., Jakobsson, M. and Juels, A. (2002) 'Optimistic mixing for exit-polls', in *ASIACRYPT '02: Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security*, pp.451–465, Springer-Verlag, London, UK.

Herrnson, P.S., Niemi, R.G., Hanmer, M.J., Bederson, B.B. and Conrad, F.C. (2008) *Voting Technology: The Not-so-Simple Act of Casting a Ballot*, in Brookings Institution Press.

Jakobsson, M., Juels, A. and Rivest, R.L. (2002) 'Making mix nets robust for electronic voting by randomized partial checking', in *Proceedings of the 11th USENIX Security Symposium*, pp.339–353, USENIX Association, Berkeley, CA, USA.

Juels, A., Catalano, D. and Jakobsson, M. (2005) 'Coercion-resistant electronic elections', in *WPES '05: Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, pp.61–70, ACM, New York, NY, USA.

Lebre, A.Z.R., Joaquim, R. and Ferreira, P. (2004) 'Internet voting: improving resistance to malicious servers in revs', in *Proceedings of IADIS International Conference on Applied Computing*.

Lundin, D. (2008) 'Component based electronic voting systems', in Chaum, D., Kutylowski, M., Rivest, R.L. and Ryan, P.Y.A. (Eds.): *Frontiers of Electronic Voting, ser. Dagstuhl Seminar Proceedings*, No. 07311, Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum f¨ur Informatik (IBFI), Schloss Dagstuhl, Germany, available at http://drops.dagstuhl.de/opus/volltexte/2008/1300.

Neff, C.A. (2004) 'Verifiable mixing (shuffling) of elgamal pairs', in *Workshop on Privacy Enhancing Technologies '03*.

Park, C., Itoh, K. and Kurosawa, K. (1994) 'Efficient anonymous channel and all/nothing election scheme', in *EUROCRYPT '93: Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, pp.248–259, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Subramonian, V., Deng, G., Gill, C., Balasubramanian, J., Shen, L., Otte, W., Schmidt, D., Gokhale, A. and Wang, N. (2007) 'The design and performance of component middleware for qos-enabled deployment and configuration of dre systems', *Journal of Systems and Software*, Vol. 80, No. 5.