Panos Louridas—Ed. (GRNET)
George Tsoukalas (GRNET)
Dimitris Mitropoulos (GRNET)

Final Service

Deliverable D5.4

January 31, 2019
PANORAMIX Project, # 653497, Horizon 2020
http://www.panoramix-project.eu

Dissemination Level: Public

Revision History

Revision	Date	Author(s)	Description
0.1	2018-10-20	DM, GT, PL (GRNET)	Proposed table of contents
0.2	2018-11-20	DM, GT, PL (GRNET)	Added evaluation diagrams
0.3	2018-11-21	DM, GT, PL (GRNET)	Added PANORAMIX Trust Control Panel
0.4	2018-11-22	DM, GT, PL (GRNET)	Added evaluation chapter & requirements
0.5	2018-11-25	DM, GT, PL (GRNET)	Added intro draft
0.6	2018-12-05	DM, GT, PL (GRNET)	Added architecture part
0.7	2018-12-10	DM, GT, PL (GRNET)	Added conclusions & minor corrections
0.8	2018-12-15	TZ (UEDIN)	Comments on the deliverable
0.9	2018-12-17	DM, GT, PL (GRNET)	Addressed comments from UEDIN
1.0	2018-12-17	MW (UEDIN)	Final pass & draft submitted to EC
1.1	2019-01-31	MW (UEDIN)	Final version submitted to EC

Executive Summary

We describe the final version of the Zeus e-voting system, which was developed in the context of Work Package 5. It is an integrating service, where Zeus is running in the context of PANORAMIX and can work with different mix-nets to carry out an electronic election. The final service incorporates our adjustments following the experience gained with the PANORAMIX MVP (described in Deliverable 5.2) and the integrated service (discussed in Deliverable 5.3). We have evaluated the final version by integrating two different mix-nets and creating testbeds to observe how the service handles hundreds of thousands of votes, with very positive results.

Contents

Executive Summary **5**

1 Introduction **9**

2 Fulfilled Requirements **11**
2.1 E-voting Context Roles 11
2.2 Authentication and Key Management 11
2.3 Usable Election Configuration 12
2.4 Usable Mix-net Configuration 12
2.5 Integrated Mix-nets 12

3 PANORAMIX Integration with Zeus **13**
3.1 Overall Architecture 13
3.2 Integration with PANORAMIX Services 14
3.3 Mix-net and Trustees 14
3.4 Architectural Specification of a Zeus Election Process 16
3.4.1 Software Integration 16
3.4.2 Architectural Specification Walkthrough 17

4 Integrated mix-nets and Evaluation **21**
4.1 The Hat-shuffle Mix-Net 21
4.2 The Verificatum Mix-net 22

5 Conclusions **25**

A Zeus Configuration Integration Reference **29**

1. Introduction

We present the current, updated version of Zeus [8], an e-voting application developed by GRNET. By employing the technology and the tools developed in the context of the PANORAMIX project, GRNET has extensively enhanced the security, performance, and usability of the application, as we describe in this deliverable.

Zeus has been in production since late 2012. Figure 1.1 illustrates the registered users and the actual voters for all the elections that took place from early 2014 to this day. In general, voters were fewer than the users that were initially registered in the electoral roll, which is consistent with the norms observed in paper-based ballots. As the figure shows, Zeus has been in continuous use and production; this means that any improvements developed inside PANORAMIX should be at a production-level maturity, and not just proof of concept implementations.

As we described in the project's proposal and in Deliverable 5.1, a limiting factor in the wider adoption of e-voting is the mix-net implementation used by the voting platform. Specifically, the Zeus e-voting system used a standard mix-net implementation that is simple, but not very efficient; holding elections with more than a few thousand voters cannot produce results in an acceptable amount of time (about an hour). Following PANORAMIX, the updated version of Zeus can run with four different mix-nets and can handle efficiently hundreds of thousands of votes, as we extensively describe in our evaluation.

The reminder of this deliverable is as follows: first, we provide the basic requirements that are met by the final service in Chapter 2. In Chapter 3 we describe in detail the architecture of the final system, and in Chapter 4 we discuss its evaluation. Finally, in Chapter 5 we provide our conclusions and lessons learned.

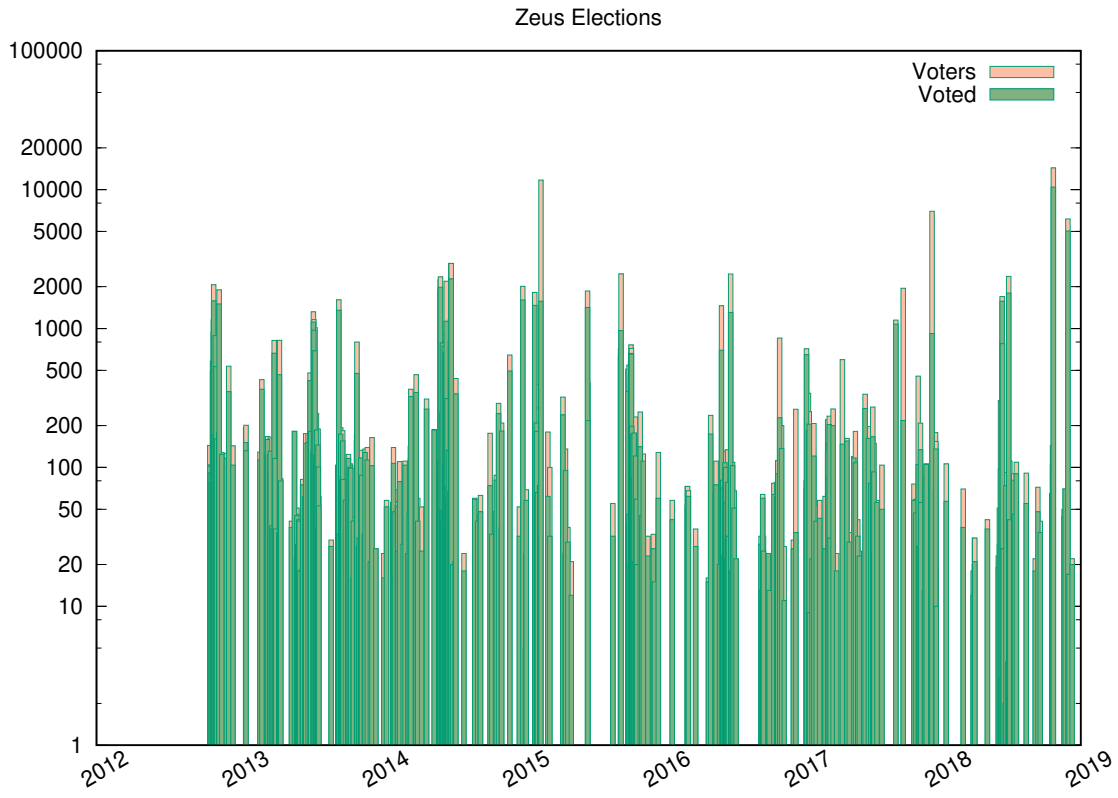

Figure 1.1: Zeus elections from 2012 to the present.

2. Fulfilled Requirements

We describe the requirements that were considered critical (as stated in Deliverable 5.3 and 4.3) and were met in the context of PANORAMIX. Note that, as we have described in Deliverable 5.1, the Zeus workflow consists of 7 basic steps, namely: (1) registration, (2) generation of the trustee keys, (3) voter invitation, (4) voting, (5) mixing, (6) decryption and (7) generation of the results, transcripts and proofs. Currently, the PANORAMIX services support the majority of the above steps with new functionalities and features, thus fulfilling all the requirements of the use case.

Moreover, the final service utilizes all three PANORAMIX services:

- The registration service is used to introduce, authorize, and identify participants.
- The configuration service is employed to produce service deployment parameters that cover trustees and mixers.
- Finally, the messaging service to mix votes in a secure manner.

2.1 E-voting Context Roles

The two basic roles in the e-voting context include *trustees* and *mixers*. Each role has specific requirements and expectations that the final service meets successfully. First, each trustee launches the PANORAMIX software to generate, keep, and contribute a secret share to the election encryption key. Then, trustees validate the shares of other trustees and agree on basic parameters for the election including (1) name, (2) ballot type and content, (3) date, (4) voter list and (5) mixer list. Furthermore, they propose and consent to start the election process using the agreed parameters. Finally, they validate mixing and they decrypt the mixed ballots.

Turning to the mixers, each one of them utilizes the PANORAMIX software to agree on the mixing algorithm and the corresponding parameters, receive votes and shuffle them and optionally validate other mixers' shuffles.

2.2 Authentication and Key Management

The final service has means to authenticate the different entities (trustees, mixers, and voters) by employing public-key cryptography through the *registration* service. Furthermore, all entities in the mix-net produce their identity credentials in the form of a public-private key pair and record it together with a mix-net specific role designation (e.g., trustee). Then, the registration service decides on and distributes a common view of the roles and peers in the network. The generation of identities and the assignment of roles are mix-net specific actions, and may involve different configuration steps that utilize the service as we discuss in the next section.

2.3 Usable Election Configuration

The configuration service provides means to set-up a secure configuration of an election in an easy and usable way. Trustees can agree on specific common preferences and parameters and record the agreements to support the auditability and accountability of the mix-net operations. The configuration service accepts various proposals for an agreement and then trustees negotiate through rounds where they collect the proposals of other administrators and submit their own. Finally, the service publishes the agreements in a secure and immutable manner. Notably, the agreements can be recorded on a secure bulletin board as described in the Deliverable 2.10 and on the corresponding research work done in the context of the project [7].

2.4 Usable Mix-net Configuration

PANORAMIX provides a usable and intuitive user interface that presents all decisions and actions in a unified way so that the mixers can have sufficient overview themselves without needing to delegate to more technically savvy operators. Besides, mix-net contributors should be independent because they have critical responsibilities for the process. At the same time, non-experts are hesitant to take on such responsibilities, so the provision of a straightfoward process and service is essential. The final service is both usable and inspires safety that address those issues.

2.5 Integrated Mix-nets

One of the basic requirements of the PANORAMIX system is to support mix-nets with specification diversity so that a wide range of use-case scenarios can be covered. Currently, our system supports a number of different mix-nets, including:

1. the *Sako-Kilian re-encryption* mix-net [5] (i.e., the mix-net that Zeus used before the project started),
2. the *Bayer Groth* mix-net [3],
3. the Verificatum Mix-Net (VMN) [2], and
4. a prototype mix-net based on the re-encryption mix-net designed by Fauzi et al. [6] (known as hat-shuffle).

The first two mix-nets have been already discussed in Deliverable 5.3. The third mix-net is a new version of the hat-shuffle mix-net that has been developed in the context of PANORAMIX and the fourth is a well-known mix-net for e-voting, widely respected by the cryptographic e-voting community. The integration of Verificatum and the *Bayer Groth* mix-net are examples of how mix-nets designed outside of the project can be easily incorporated into our service (which, in the big picture, is essential for standardization, as we discuss in Deliverable 4.4).

3. PANORAMIX Integration with Zeus

We describe the complete integration of the Zeus application with PANORAMIX. First, we describe an overall architecture to link the chapter to Deliverable 5.3. Then we discuss specifics regarding the integration with the different PANORAMIX services and show the architectural specification of an election process in the context of the final service.

3.1 Overall Architecture

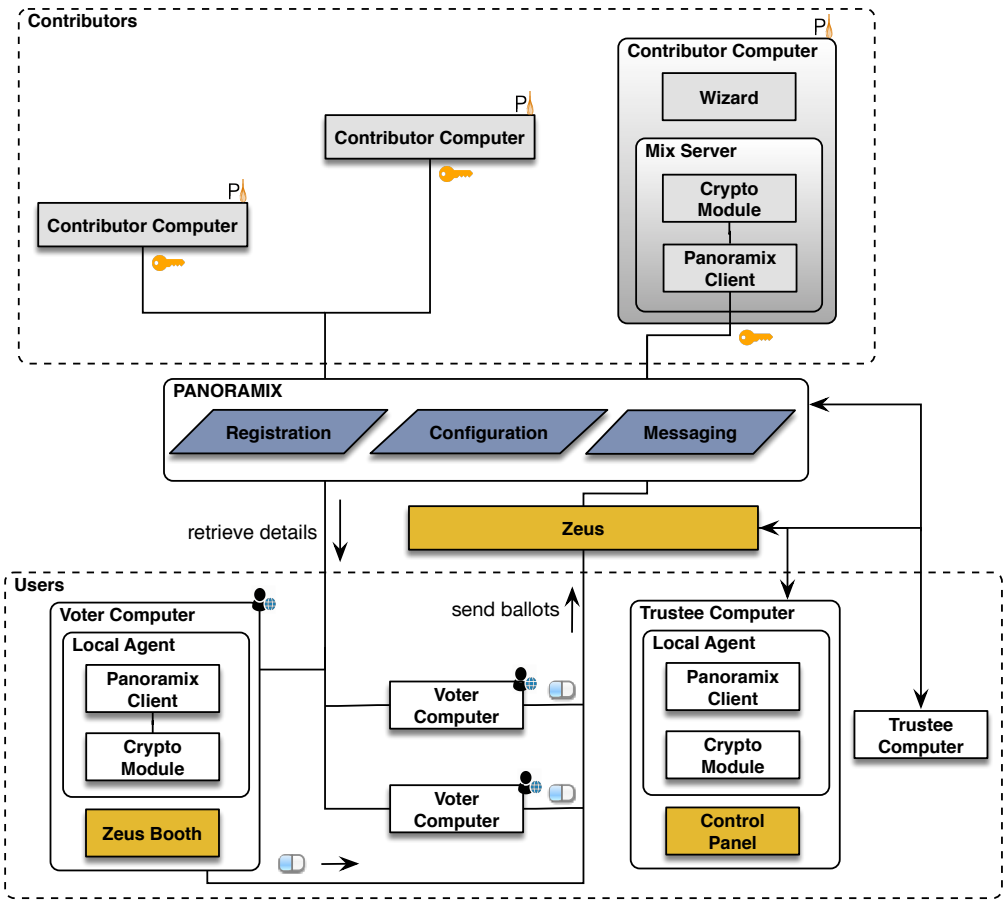

Figure 3.1: The architecture of the final service.

We provide a description of the overall architecture of the final service, based upon the PANORAMIX software toolkit. Specifically, we describe how Zeus uses the three PANORAMIX services, namely: registration, configuration, and messaging. An overview is depicted in Figure 3.1, and it involves the controller system with the three services, voters, trustees, mix-net

contributors and their interaction. More details regarding the overall architecture can be found in Deliverable 5.3.

3.2 Integration with PANORAMIX Services

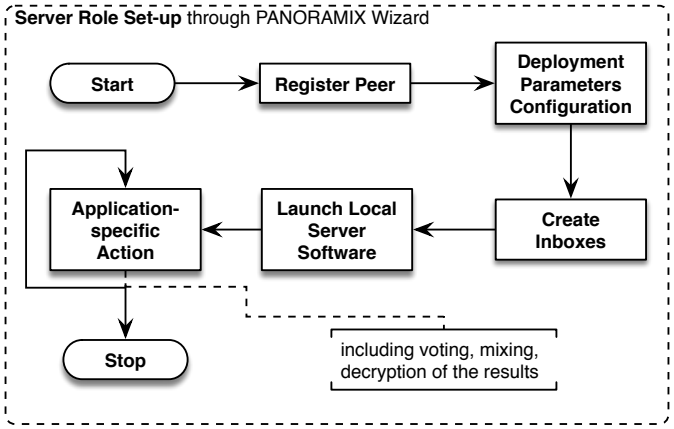

Figure 3.2: The workflow behind the set-up of a server role through the PANORAMIX wizard.

First, Zeus uses the registration service to manage public key identities for

- any PANORAMIX component (e.g., controller),
- Zeus trustees and mixers,
- any authenticated servers operating under trustee or mixer control.

Voters, authorities, and auditors may query the registration service to retrieve details for the identities of participants whose keys appear in the proofs Zeus compiles for the whole election process.

To use the PANORAMIX tools through the wizard for configuration and setup, the integration framework must provide configuration options and corresponding scripts that can be used to launch software and act on specific choices. Figure 3.2 displays the steps that a user must follow to set-up a server through the PANORAMIX wizard. First, the registration credentials must be provided. Then, the deployment parameters are configured. The user should also create the inboxes (used for forwarding mix-net payloads) and launch the software. Finally, there are actions that are use-case related. Hence, the finally step could take place multiple times.

When the Zeus reaches the voting stage, an election inbox is launched, where PANORAMIX clients may submit votes through the generic PANORAMIX messaging API described in Deliverable 5.3.

3.3 Mix-net and Trustees

Zeus e-voting is an application essentially built around a mix-net. The mix-net in Zeus provides the most technologically challenging aspect of the system, the one that uses advanced cryptography for the anonymization of submitted encrypted ballots along with mathematical proofs of correctness that hide the actual shuffling (zero-knowledge).

The mix-net is composed of a sequence of independent servers, the mixers, each one of which secretly shuffles the ballots of the previous mixer. The input of the first mixer is the contents of

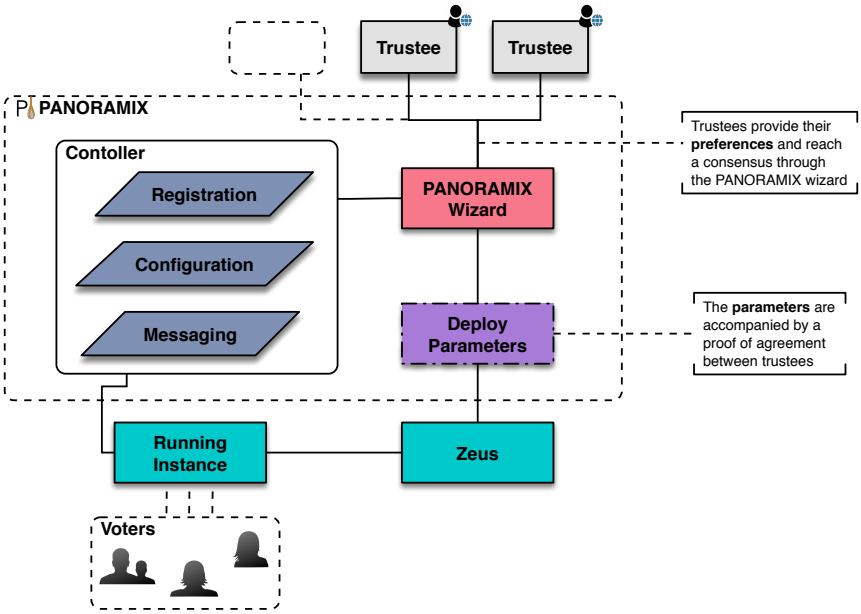

Figure 3.3: Trustees using the PANORAMIX Trust Control Panel.

the ballot box as it was collected throughout the voting process. The output of the last mixer is the ballots to be decrypted in order to obtain the final results.

Each mixer provides a proof that they have performed the shuffle correctly, and that in doing so they have preserved the original contents of the ballots. Therefore, we do not have to trust any of them for correctness. However, the shuffle is only effective if the mixer keeps the shuffle details secret. Therefore, we have to trust that at least one of the mixers is honest, which is the basic security requirement by Zeus: anonymity is guaranteed as long as at least one of the mixers is honest, even if all the other misbehave.

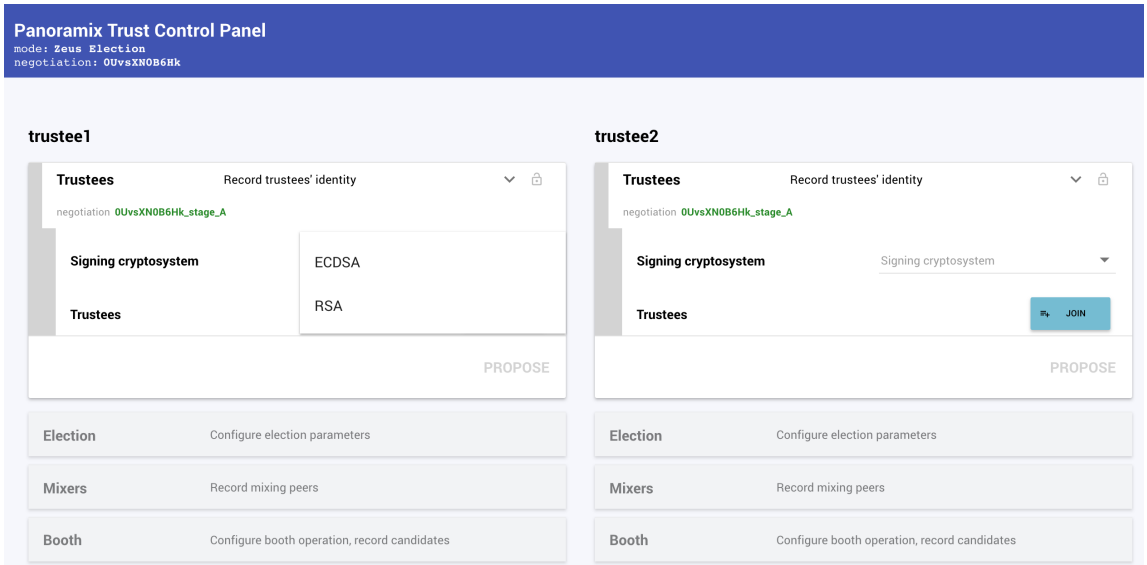

Figure 3.4: PANORAMIX Trust Control Panel: Trustees record their identities.

Moreover, the input of the mix-net has to be encrypted, and the output decrypted. The cryptographic operation of verifiable shuffling dictates the cryptography used for the encryption of the ballots and their decryption after mixing. The cryptographic shuffles can only shuffle ci-

phertexts from a specific cryptosystem. Therefore, the trustees must use the same cryptosystem, with the same parameters, as the mixers.

Additionally, trustees share a similar trust requirement as the mixers. Trustees provide cryptographic zero-knowledge proof for their ownership of their key shares and the correctness of their decryption. However, if the trustees decrypt anything other than the mixed ballots, or they do not check the shuffle proofs before decrypting, the secrecy of the election is compromised. Therefore, we must trust that at least one of the trustees is honest; or, in other words, Zeus guarantees correct decryption as long as at least one of the trustees is honest.

Because of the similar trust requirements for both trustees and mixers, the two roles can be combined so that the need to trust is reduced. It is safer to have to trust at least one of the trustee-mixers than it is to trust at least one trustee *and* at least one mixer. Having to trust both trustees and mixers is a stronger security assumption.

Following the above, as a design choice to enhance security using PANORAMIX, we have fused the roles of trustee and mixer, at least from the perspective of the controlling end-user, providing a streamlined, user-friendly experience that can help non-experts configure their election while guaranteeing security.

The practical details of this architectural choice are presented in Subsection 3.4, where the election process has been mapped in a sequence of configuration choices across different stages.

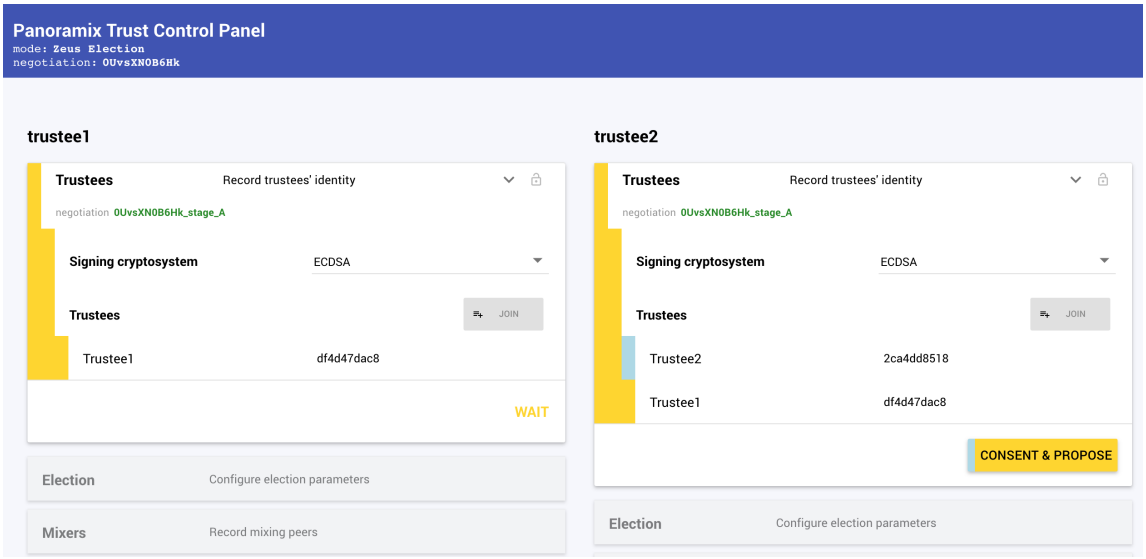

Figure 3.5: PANORAMIX Trust Control Panel: The first trustee proposes a specific signing cryptosystem (ECDSA – Elliptic Curve Digital Signature Algorithm).

3.4 Architectural Specification of a Zeus Election Process

In this section we discuss details on the software integration and provide an architectural specification walkthrough of a trustee.

3.4.1 Software Integration

The "Trust Control Panel" depicted in Figures 3.4 to 3.8 is a user-friendly name for the PANORAMIX wizard after it has been integrated and packaged as a Zeus component. The PANORAMIX framework allows applications to provide specifications for the configuration options that have to be presented to the persons that control important parts of the process (such as the mixer servers, or additionally in the case of Zeus, the trustees).

All parties must agree on the exact same parameters that are then enacted by the application. The PANORAMIX wizard consists of a back-end, the PANORAMIX local agent as was described since the first versions of the architecture, and a user-friendly front-end that is controlled by the back-end agent. This provides flexibility to either deploy the wizard locally at the trustee's computer, or deploy the wizard at a remote trusted server, while retaining the ability to control it through a graphical user interface.

The screen-shots in the figures are from a Trust Control Panel user interface that has been integrated for a demonstrator election.

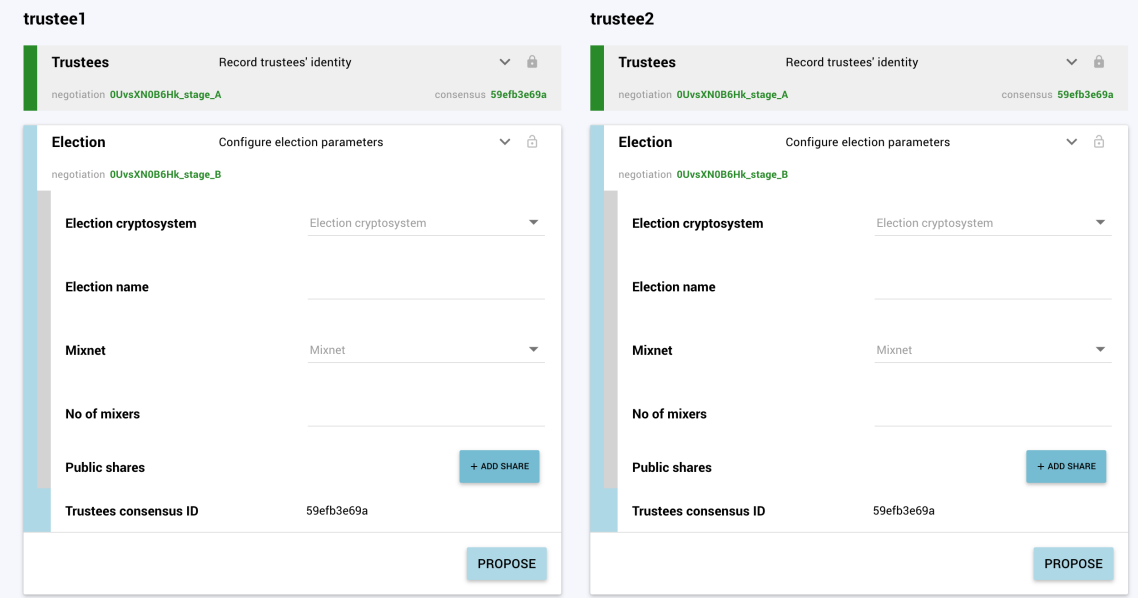

Figure 3.6: PANORAMIX Trust Control Panel: The trustees move on to configure the election parameters.

3.4.2 Architectural Specification Walkthrough

In this section we describe the specification for the Trust Control Panel of a Zeus trustee. This specification corresponds to the important architectural parts from the trustee point of view, that is, from the view point of the important choices and agreements that have to be made by distributed independent participants in order to guarantee proper election process. The reader may either browse through the screen-shots or the configuration integration reference document that is listed in the next section. Note that we also provide technical details regarding specifications in the Appendix.

The election process is split into seven stages, A through G:

Stage A: General cryptography setup and trustee registration (Figures 3.4, 3.5).

Trustees, mixers, and other important actors in the process must be authenticated by their cryptographically secure signature for every important action or communication they make. The cryptographic setup for this functionality does not depend on the actual cryptography used for encrypting, mixing, and decrypting votes. Nevertheless, it is an important security configuration and participants must be able to inspect and ensure that strong enough technology is being used according to their specific use-case need.

Besides setting up this general cryptography setup, the identities of the trustees themselves have to be registered as the controlling actors of every important choice in the following steps.

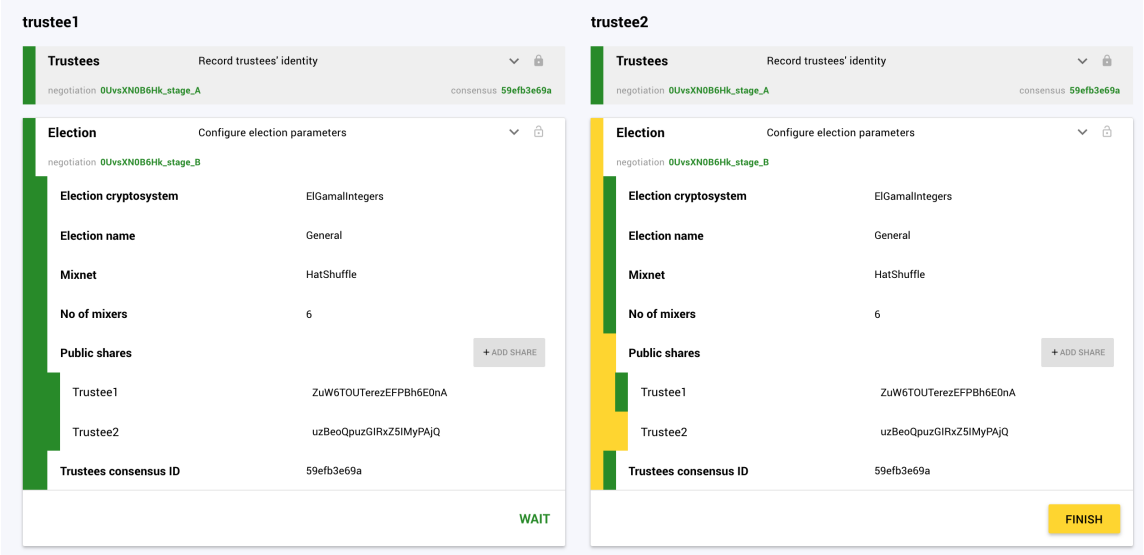

Figure 3.7: PANORAMIX Trust Control Panel: The Trustees indicate if they are going to take part on the mixing process.

Stage B: Setup election cryptography, trustee-shared election key, type of mixnet (Figures 3.6, 3.7).

The executive power distributed to trustees is possible through the cryptographic technology of secret sharing, applied on the private key of the election. In effect, this means that each trustee controls part of the secret key of the electoral process. The public key of the election is a combination of the distributed secret private keys. All votes are encrypted using the combined election public key. The decryption of any vote requires the consent and action of every trustee to use their secret to produce partial decryptions. Using this power, any of the trustee can technologically prevent the election process from producing any result whatsoever. Therefore, the production of the election key in this stage is one of the most important trust acts in the process.

For mathematical reasons, the cryptosystem used to create the election public key must be exactly the same as the cryptosystem used by the mixnet. Therefore the mixnet type must be decided in this stage along with the election key.

Stage C: Mixer registration (Figure 3.8).

Technically, the set of mixers can be independent from the set of the trustees. However, as analyzed previously, to minimize the trust assumptions and related risks, the ideal configuration is to assign trustee and mixer roles to the same group of people.

At this stage, the identities of the people responsible for the independent mixers have to be registered so that they can be known for the rest of the process, in order to be accountable, and so that they can acquire the necessary executive power to contribute to the election process.

Stage D: Election ballot, poll event setup, voter registration.

This stage defines all the election attributes that are important for the application semantics of the actual election process. The record of these choices is critical for the interpretation of the results and for every auditing that is made to verify them, or to investigate any incident or anomaly.

Stage E: Voting.

During this stage votes are received by the ballot box. The important outcome for this stage is a strong commitment to the exact contents of the ballot box to be processed. Again, record of this information is essential for the verification of results or for investigating by auditors.

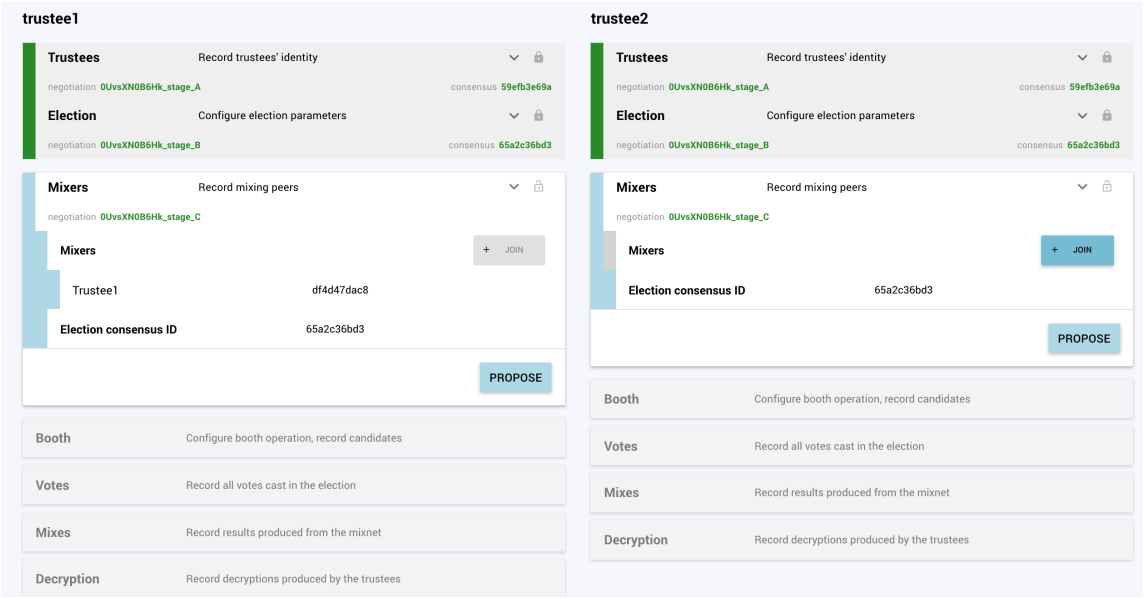

Figure 3.8: PANORAMIX Trust Control Panel: The trustees move on to configure the election parameters.

Stage F: Mixing.

Encrypted ballots go through a pipeline of mixers that shuffle their input and forward the result for further mixing along with cryptographic proofs of correctness. Mixers, just as trustees, are responsible for the secrecy of the votes. Therefore, it is important for them to verify the entire election process until their input, including the shuffle proofs of preceding mixers. Failure to do that may lead to them not actually participating in the election process they have committed to protect.

Stage G: Decryption of results.

In this final stage, the trustees must make the ultimate verification of everything up to this point before they use their secret to enable decryption of the results. Failure to verify means, (a) that they may be led to decrypt ciphertexts that have not been mixed and therefore their secrecy is not protected by any mixers, or (b) that a mixer manufactured their own encrypted ballots in replacement of the legitimate ones, therefore compromising the entire election result. It is critical that trustees do not allow their secret key to be used in unverified, untrusted ballots.

4. Integrated mix-nets and Evaluation

To evaluate the final service we used two mix-nets: (a) the Verificatum Mix-Net (VMN) [2], (b) a prototype mix-net based on the re-encryption mix-net designed by Fauzi et al. [6] (also known as hat-shuffle). We have also developed test suites to validate the functionality of the service.

We describe the integration of the two mix-nets and the corresponding benchmarking we performed after their integration to evaluate their efficiency. Note that these are the latest mix-nets that were integrated, which is why we focus on them here. Recall that we have previously also ported the Sako-Kilian re-encryption mix-net [5] and the Bayer Groth mix-net (as discussed in D5.3).

4.1 The Hat-shuffle Mix-Net

The hat-shuffle mix-net was developed in the context of PANORAMIX [1]. Specifically, it is a non-interactive zero-knowledge proof (NIZK) [4] shuffle argument. We discussed the initial integration in Deliverable 5.3. Since then, various software updates have taken place, hence we provide the evaluation done with the latest version of the mix-net.

One of the key components of this mix-net involves a common reference string (CRS). Specifically, a CRS model incorporates the assumption of a trusted setup in which all involved entities get access to the same string CRS taken from some distribution D. Schemes proven secure in the CRS model are secure given that the setup was performed correctly, which actually happens in PANORAMIX as we have already explained.

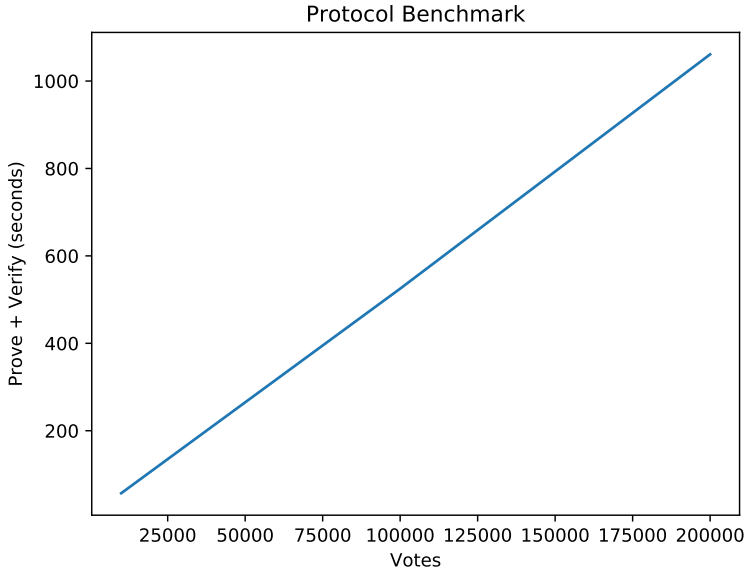

Figure 4.1: Relationship between number of votes and time.

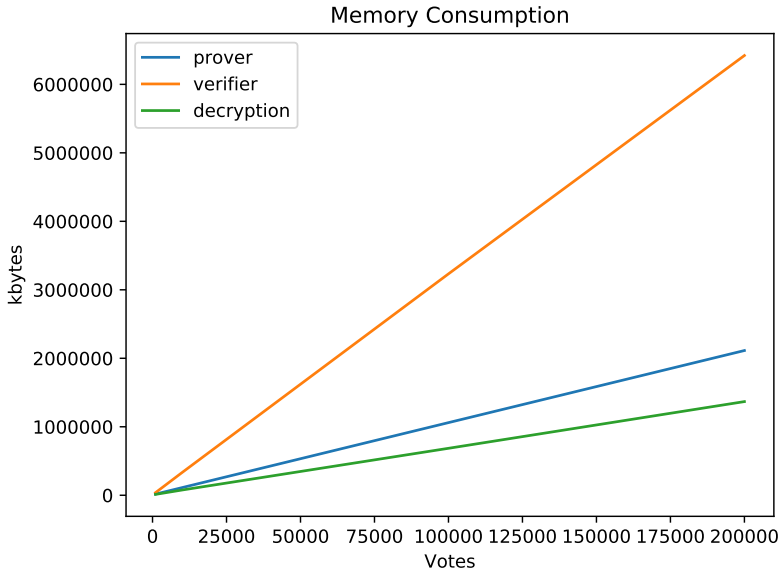

Figure 4.2: Relationship between number of votes and memory.

We have performed extensive tests regarding the performance of the hat-shuffle integration. More specifically, we have examined the generation and serialization of the CRS. In addition, we have written benchmarks for all the phases of the mix-net including (1) encryption, (2) proving, (3) verification and (4) decryption. Our benchmarks included a different number of votes each time. We started with 10K votes, continued with 100K, then 200K and finally 1M votes. Our tests included settings where threads were running in parallel. Our experiments were performed on a run-of-the-mill Ubuntu machine (16.04) with 4 CPUs, 8GB RAM and 40GB system disk.

Figure 4.1 presents the different sizes of the files that are generated, for the different numbers of votes. We observe that there is a linear correlation and as the message number becomes larger the size increases too. This linearity is important because it proves that the theoretical scalability of the algorithm is indeed achieved in practice.

A linear relationship also exists between memory consumption and number of votes. Figure 4.2 illustrates the relationship of these two. To measure the relationship we used the same methodology as the one described above, specifically, measuring the memory consumption for a different number of votes.

We need to highlight here that hat-shuffle handles 1M votes efficiently as seen in Table 4.1. Recall that Zeus (the e-voting platform involved in WP5), uses a standard mix-net implementation that is simple, but not very efficient; only a few thousand voters can be handled within an acceptable amount of time (about an hour). Now, by using hat-shuffle through PANORAMIX, Zeus will be able to support millions of voters as set out in the beginning of the project.

4.2 The Verificatum Mix-net

Verificatum is an open-source re-encryption mix-net written in Java that can be employed in online voting. As we mentioned in Chapter 2, the integration of Verificatum is an example of how a mix-net designed outside the PANORAMIX project can be used effectively on the PANORAMIX platform.

A typical Verificatum workflow is that the operators of the mixing nodes have a physical meeting where they agree upon common parameter files and then employ them on their respective servers during deployment. This is no longer necessary in the context of our final

Table 4.1: Hat shuffle mix-net benchmark

module	phase	1.000.000 votes
crs	Generation	218.1849s
	Serialization	152.4044s
	Total	370.5893s
generate_votes		66.1600s
encrypt	CRS Deserialization	31.2038s
	Votes Deserialization	4.0429s
	Encryption	2700.1721s
	Ciphertexts Serialization	50.7118s
	Total	2786.1306s
prove	CRS Deserialization	56.2043s
	Ciphertexts Deserialization	45.9318s
	Prove	389.7539s
	Proofs Serialization	101.1021s
	Total	603.2646s
verify	CRS Deserialization	58.4569s
	Ciphertexts Deserialization	42.6796s
	Proofs Deserialization	103.6393s
	Verify	1655.6244s
	Total	1860.4001s
decrypt	Votes Deserialization	2.0494s
	Ciphertexts Deserialization	36.8059s
	Table	408.1059s
	Decryption	381.1743s
	Votes Serialization	0.5766s
	Total	828.7121s
mix *		41:03m

* mix = Prove + Verify
** MacOSX: 16GB RAM + SWAP Memory
-P: Running Parallel Threads
VM specs: Ubuntu LTS 16.04 (image), 4 (CPUs), 8192MB (RAM), 40GB (System Disk)

service because of the functionalities provided by the PANORAMIX Trust Control Panel, which we described earlier.

The Verificatum servers automatically create shared keys for encryption and decryption of ciphertexts, proceed with the mixing of ciphertexts, check each other's mathematical proofs, and make the final results available locally to each node administrator. Nevertheless, to integrate Verificatum into a generic PANORAMIX framework we had to isolate the actual mixing phase from the typical workflow of the mix-net. In particular, the encryption keys are not generated by Verificatum itself but they are part of the initial deployment parameters agreed upon by all contributors. In addition, Verificatum has its own communication system to coordinate all mixnet servers at runtime. This offers PANORAMIX no control over the workflow. Instead, Panoramix only deploys Verificatum as multiple single-node mix-nets, and uses its proof checker to link and verify the mix contributions into a proper mix-net. Since encryption keys are not generated and managed by Verificatum, Verificatum is asked to skip decryption, which is handled separately by PANORAMIX.

We have evaluated Verificatum based on the testbed that we used for hat-shuffle. Table 4.2 shows the efficiency of Verificatum and its functionalities for different numbers of votes. Again, it is clear that the number of votes that can be anonymized has reached the million point threshold.

votes	initialization	mix	ciphertexts	verify
1000000	00:54	18:42	01:32	18:42
500000	00:27	08:04	00:43	08:04
100000	00:08	01:38	00:10	01:38
50000	00:06	00:53	00:06	00:53
10000	00:03	00:17	00:02	00:17
5000	00:03	00:12	00:02	00:12
1000	00:02	00:07	00:00	00:07
10000	00:04	00:43	00:02	00:43

Table 4.2: Verificatum efficiency benchmarks.

5. Conclusions

In this deliverable, we discussed the final version of the integrated service developed in the context of Work Package 5 and described in Deliverables 5.2 and 5.3. We presented the integration details of the PANORAMIX framework, including the refactoring of the Zeus process to allow specification of the critical configuration parameters that have to be agreed by trustees through the PANORAMIX tools to achieve a user-friendly but secure way for non-experts to control and safeguard elections. At the core of the election process, we have integrated a selection of multiple mix-nets, and we have tested and validated those mix-nets in the context of the election process.

In this final round of development, we worked out finer but important details such as a user-friendly visual representation of the distributed control, negotiation, and consensus that the independent trustees are responsible to exercise. These extra steps of process specification and end-user engagement make the end-to-end verifiability of the process more practical and less theoretical to the non-expert end-user. Therefore, along with the scalability of the mixing that was achieved by integrating Verificatum [2], and PANORAMIX-developed Hat-Shuffle [1], we have made solid progress to achieve our initial goal to make the process more robust and dependable so that end-users do not hesitate to take on the role of mixer or trustee, fearing neither technical incompetence, nor unmanageable responsibility, while being able to scale to elections involving millions of voters. Notably, our benchmarking in the order of millions reflects production conditions, as promised in the grant agreement for D5.4.

Bibliography

[1] Hat-shuffle implementation. https://github.com/grnet/hat_shuffle, 2018. [Online; accessed 15-September-2018].

[2] The verificatum mix-net (vmn). https://www.verificatum.com/, 2018. [Online; accessed 15-September-2018].

[3] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'92, pages 263–280, Berlin, Heidelberg, 2012. Springer-Verlag.

[4] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 103–112, New York, NY, USA, 1988. ACM.

[5] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

[6] Prastudy Fauzi, Helger Lipmaa, and Michal Zajac. An efficient pairing-based shuffle argument. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Application of Cryptology and Information Security, 2017*, 2017.

[7] Aggelos Kiayias, Annabell Kuldmaa, Helger Lipmaa, Janno Siim, and Thomas Zacharias. On the security properties of e-voting bulletin boards. *IACR Cryptology ePrint Archive*, 2018:567, 2018.

[8] Georgios Tsoukalas, Kostas Papadimitriou, Panos Louridas, and Panayiotis Tsanakas. From Helios to Zeus. In *2013 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '13, Washington, D.C., USA, August 12-13*, 2013.

A. Zeus Configuration Integration Reference

We provide the specifications that correspond to the configuration part and the stages discussed in Subsection 3.4.

```
TYPES = ['string', 'int', 'datetime', 'dict']
ACTIONS = ['choices', 'compute_element', 'compute']
MODES = ['interactive', 'auto']


TRUSTEE_OPTIONS = {
    'stage_A': {
        'signing_cryptosystem': {
            'type': 'string',
            'action': 'choices',
            'choices': ['ECDSA', 'RSA'],
        },
        'trustees': {
            'type': 'dict',
            'action': 'compute_element',
            'label': 'Join',
            'function': 'set_signing_crypto',
            'undo': 'unregister_crypto',
            'params': ['signing_cryptosystem'],
            'icon': 'playlist_add'
        },
    },

    'stage_B': {
        'stage_A_result': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'get_stage_result',
            'params': ['stage'],
            'actuals': {'stage': 'stage_A'},
            'key_label': 'Trustees consensus ID',
        },
        'election_cryptosystem': {
            'type': 'string',
            'action': 'choices',
            'choices': ['ElGamalIntegers', 'ElGamalCurves'],
        },
        'public_shares': {
            'type': 'dict',
            'action': 'compute_element',
            'label': 'Add Share',
            'function': 'set_election_public',
            'undo': 'unset_election_public',
            'params': ['election_cryptosystem'],
        },
        'election_name': {'type': 'string'},
        'no_of_mixers': {'type': 'int'},
        'mixnet': {
            'type': 'string',
            'action': 'choices',
```

```
            'choices': ['SakoKilian', 'HatShuffle', 'Verificatum'],
        },
    },

    'stage_C': {
        'stage_B_result': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'get_stage_result',
            'params': ['stage'],
            'actuals': {'stage': 'stage_B'},
            'key_label': 'Election consensus ID',
        },
        'mixers': {
            'type': 'dict',
            'action': 'compute_element',
            'label': 'Join',
            'function': 'get_signing_key',
            'params': [],
        },
    },

    'stage_D': {
        'stage_C_result': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'get_stage_result',
            'params': ['stage'],
            'actuals': {'stage': 'stage_C'},
            'key_label': 'Mixers consensus ID',
        },
        'opens_at': {'type': 'datetime'},
        'closes_at': {'type': 'datetime'},
        'candidates': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'hash_candidates',
        },
        # 'voters': {
        #     'type': 'string',
        #     'action': 'compute',
        #     'mode': 'auto',
        #     'function': 'hash_voters',
        # },
    },

    'stage_E': {
        'stage_D_result': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'get_stage_result',
            'params': ['stage'],
            'actuals': {'stage': 'stage_D'},
            'key_label': 'Booth consensus ID',
        },
        'votes': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'hash_votes',
        },
    },

    'stage_F': {
        'stage_E_result': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'get_stage_result',
```

```
            'params': ['stage'],
            'actuals': {'stage': 'stage_E'},
            'key_label': 'Votes consensus ID',
        },
        'mixed_data': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'verify_all_mixes',
        },
    },

    'stage_G': {
        'stage_F_result': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'get_stage_result',
            'params': ['stage'],
            'actuals': {'stage': 'stage_F'},
            'key_label': 'Mixes consensus ID',
        },
        'decryption_factors': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'verify_all_decryption_factors',
        },
    },
}

MIXER_OPTIONS = {
    'stage_C': {
        'stage_B_result': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'get_stage_result',
            'params': ['stage'],
            'actuals': {'stage': 'stage_B'},
        },
        'mixers': {
            'type': 'dict',
            'action': 'compute_element',
            'function': 'set_signing_crypto',
            'undo': 'unregister_crypto',
            'params': ['cryptosystem', 'path'],
        },
    },

    'stage_F': {
        'stage_E_result': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'get_stage_result',
            'params': ['stage'],
            'actuals': {'stage': 'stage_E'},
        },
        'mixed_data': {
            'type': 'string',
            'action': 'compute',
            'mode': 'auto',
            'function': 'verify_all_mixes',
        },
    },

}
```