Rafael Galvez—Ed. (KUL)
Dimitris Mitropoulos (GRNET)
George Tsoukalas (GRNET)
Panos Louridas (GRNET)
Yiannis Tselekounis (UEDIN)

Final System

Deliverable D4.4

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

Dissemination Level: Public

Horizon 2020
European Union funding
for Research & Innovation

Revision History

Revision	**Date**	**Author(s)**	**Description**
0.1	2018-08-28	DM, GT (GRNET)	Proposed table of contents
0.2	2018-09-19	DM, GT (GRNET)	Proposed draft of sections
0.3	2018-10-25	DM, GT (GRNET)	Section 2 ready
0.4	2018-10-26	DM, GT (GRNET)	Subection 5.2 ready
0.5	2018-10-27	DM, GT (GRNET)	Subection 3.1 and 3.2 ready
0.6	2018-10-27	DM, GT (GRNET)	Section 3 ready
0.7	2018-10-28	DM, GT (GRNET)	Subection 4.1 and 4.3 ready
0.8	2018-10-28	DM, GT (GRNET)	Section 4 ready
0.9	2018-10-30	DM, GT (GRNET)	Section 6 ready
1.0	2018-11-16	RG (KUL)	Conclusion and executive summary ready
1.1	2018-11-19	DM, GT (GRNET)	Addressing 1st review comments
1.2	2018-11-19	DM, GT (GRNET)	Addressing 1st review comments part 2
1.3	2018-11-19	DM, GT (GRNET)	Added Panoramix Toolkit as Appendix
1.4	2018-12-10	MB (CCT)	Review and proofreading
1.5	2018-12-17	MW(UEDIN)	Final Draft submitted to EC
1.6	2018-12-24	DM, GT, PL (GRNET)	Changes in Introduction
1.7	2018-12-30	DM, GT, PL (GRNET)	Addressed more review comments
1.8	2019-01-03	AK(UEDIN)	Editorial pass
1.9	2019-01-05	RG(KUL)	Moved license to appendix
2.0	2019-01-31	MW(UEDIN)	Final version submitted to EC

Executive Summary

This deliverable describes the final system of the PANORAMIX framework, which addresses the remaining integration issues and incorporates the external feedback collected from the use-case work packages. All the use cases make use of the system to fulfill their requirements.

An internet service has been created to support their operations, and the quality of the code has been improved to enable third party developers adapt it to their needs or simply use it as a library in an effective way.

Section 2 provides an overview of the different components of the system, and section 3 details how a given application (e.g. email, e-voting) may use them.

Sections 4 and 5 describe the software and its validation process in order to make its inner workings transparent enough for third party developers to understand the system and potentially adapt it to their needs.

Contents

1. Introduction

We describe the final integrated system. A production-ready system and a corresponding Internet service that has been shown to work with the proposed use cases. The system is based on a software architecture that has been described in Deliverables 4.2 and 4.3 and has evolved to the one described in this document. In addition, the system involves a stable API and it is accompanied by documentation for (1) System Administrators and (2) Software Developers (which can be found in the Appendix of Deliverable 4.3). To support a wide variety of different use cases PANORAMIX offers two types of integration (see also Figure 1.1): (1) a "tight" integration in the case of applications and mix-nets where the network is setup and operated from the core PANORAMIX framework (such as Zeus in the e-voting use case and the Anonymization Client utilised in the WP6 use-case for statistics, cf. Deliverable 6.2), and (2) a "loose integration" where the network components operate independently from the core framework while the framework is used to setup and launch the software that connects to the network per se; such is the case with the *Katzenpost* component (developed in WP7).

The system has evolved towards an easy to use and fully featured product that third parties with different aims can easily use. The latest developments in WP5, WP6, and WP7 already utilize the system and showcase its flexibility and ease of use.

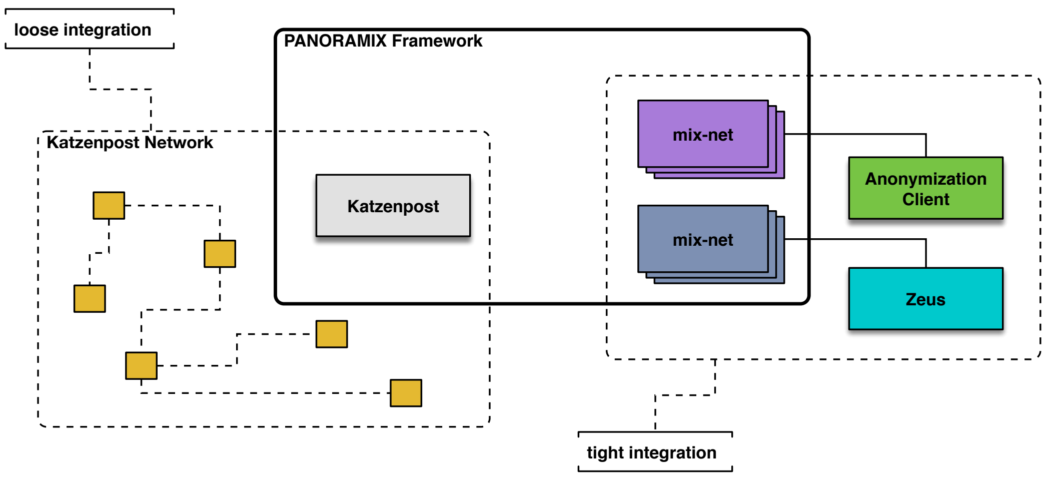

Figure 1.1: PANORAMIX Offers Two Types of Integration.

2. PANORAMIX Framework Services

2.1 Overview

In the context of the PANORAMIX system, each individual mix-net is organized around three basic (micro-)services including: the *registration service*, the *configuration service*, and the *messaging service*. Figure 2.1 illustrates the services and their high level interactions with (1) the various contributors and (2) the users.

The aforementioned services were designed and implemented to meet the requirements set in work packages 5, 6 and 7 and described in detail in Deliverables 4.2 and 4.3. First, general requirements such as peer authentication and Public Key Infrastructure (PKI), usable and secure mix-contributor configuration and audit-log for administrators, and integration between different mix-net implementations and the controller are satisfied. And second, specific use-case related requirements are met: the trustees of an e-voting process are able to control the election procedure (including mixing), while a mix server identity registry for secure messaging is facilitated.

2.2 Registration Service

The fundamental authentication element that PANORAMIX provides for participants to engage with the system is a public-private key pair. This key pair can be used by the actor to sign information such as protocol requests and responses, and by anybody else to encrypt information meant to be communicated to the actor.

In the context of PANORAMIX, the registration service provides peer authentication and a public key infrastructure that offers identity management. Figure 2.2 illustrates how the various actors interact with the service.

All entities in the mix-net should produce their identity credentials in the form of a public-private (PK) key pair and record it together with a mix-net specific role designation. Then, the service is responsible to decide on and distribute 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 Configuration Service

Most security-related applications and protocols, including mix-nets, distribute trust among network entities that are considered independent and are secured in different domains. However,

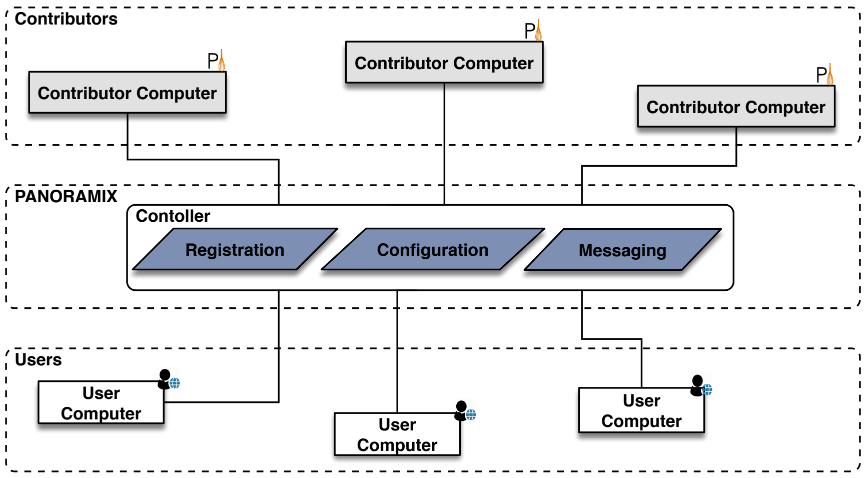

Figure 2.1: The Three Basic PANORAMIX Services.

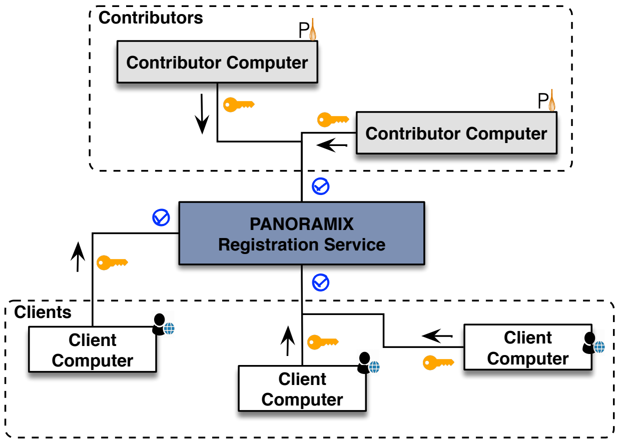

Figure 2.2: The registration service is responsible for the PKI, the authentication of the different actors and the role assignment. Specifically, it registers a role for clients and administrators and is able to authenticate a contributor or a client based on their public key.

these distributed entities need to collaborate in order to correctly execute a protocol and the coordination between them and its security implications are often overlooked.

The configuration service provides means to set-up secure configuration and auditability features. Different entities in a mix-net must agree on specific common preferences and parameters. The agreement and configuration availability is critical for the functionality and security of the mix-net. An audit record of the agreements is also important for the auditability and accountability of the mix-net operations.

The configuration service may accept various proposals for an agreement under an arbitrary title. Then, administrators 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.

The aforementioned actions can be performed in a user-friendly manner through the PANORAMIX wizard which we describe in an upcoming section.

The final agreement among all relevant participants takes the form of a single document we call the common Service Deployment Parameters. All important configuration parameters for all participants must be part of this document.

Although the production of such a document can be negotiated manually and from scratch, PANORAMIX offers tools that allow software to start from a template to guide the end-user through the important choices. This configuration template must be part of the integration of a mixnet with the PANORAMIX framework.

2.4 Messaging Service

The messaging service accepts encrypted messages for delivery through a mix-net that has been configured using the framework. The service offers a message inbox where incoming messages are queued for consumption. Messages are sent and received by end-users. Nevertheless, the messaging service may also be leveraged for internal communication among the mix-net entities to exchange information regarding the whole system.

The messaging service employs two kinds of endpoints that can be utilized by a registered client computer which in turn will send and receive encrypted messages via the mix-net. The service offers an endpoint to inform clients of the cryptographic algorithms used and an endpoint to create and use message in-boxes. In particular, the latter provides specific information regarding the kind of the mix-net and the various parameters that have to be used to either encrypt or decrypt messages. The endpoints used by the contributor computers may vary based on the protocol that is being used (e.g., decryption/re-encryption mix-net).

3. Use-case Support

In this chapter we describe how the implemented PANORAMIX services support various use-cases. As illustratory examples we describe the e-voting, e-mail and anonymised surveys and statistics use cases.

3.1 Overview

Figure 3.1 illustrates how the administrators of a specific use case can configure their software through PANORAMIX.

First, they need to login to the PANORAMIX wizard and connect to the relevant PANORAMIX controller (i.e. to the relevant registration service). Through this wizard they can provide their **preferences**, see the proposed preferences of other users and decide upon an **agreement** regarding the **parameters** of the running software. These parameters are related to both the functionality and the security of the running software.

When the administrators reach a consensus, PANORAMIX records the agreement and stores the corresponding **proof**. Then, it automatically deploys the system according to the agreed parameters, which in turn interacts with the framework and its different services.

3.2 Workflow

The general workflow for setting up a mixing service with PANORAMIX tools starts with the administrators of each of the server nodes that participate in the service using the PANORAMIX wizard and the related tools (see the sysadmin manual Appendix). These tools enable administrators to configure and deploy the software server of their local nodes in coordination with the other peers. It is possible that there are more than one type of server nodes, and therefore their administrators assumes different roles.

For instance, in the case of a re-encryption mix-net for elections, there are two roles: trustees and mixers. Both roles are implemented by two different pieces of software running in the server. On the one hand, trustee nodes hold the election key shares and can decrypt the mixed ballots. On the other hand, mixer nodes provide the actual mixing and proofs. The server node administrators must specify which role they want to deploy software for.

Each server node administrators follow the following steps:

1. Install the PANORAMIX toolkit (including the PANORAMIX wizard)
2. Register the local node at the Registration Service launched by the Controller of the service they want to participate.

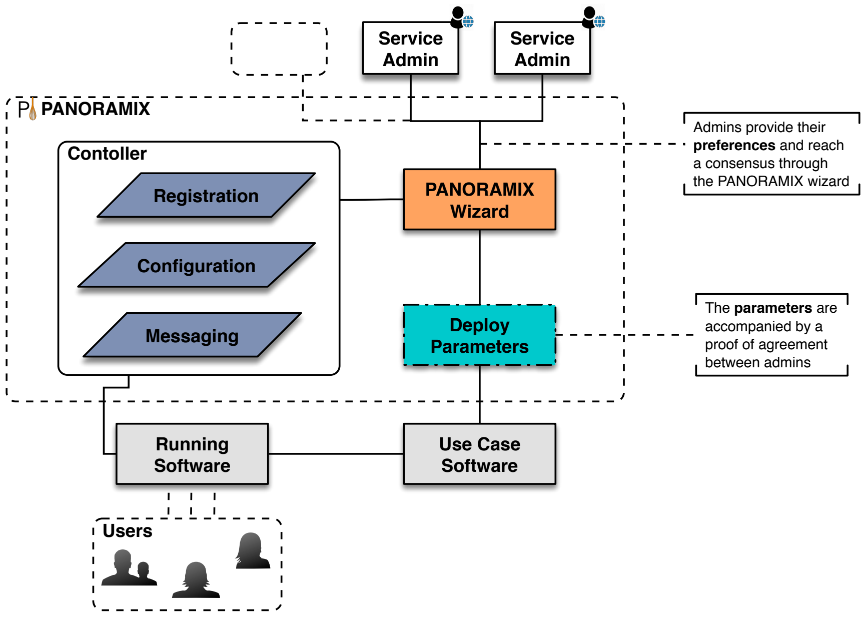
Service Admin
Service Admin
PANORAMIX
Contoller
Registration
Configuration
Messaging
PANORAMIX Wizard
Deploy Parameters
Admins provide their **preferences** and reach a consensus through the PANORAMIX wizard
The **parameters** are accompanied by a proof of agreement between admins
Running Software
Use Case Software
Users

Figure 3.1: Setting up a specific use case in our framework.

3. Negotiate the common service deployment parameters. The PANORAMIX configuration tool uses a mix-net specific template which is provided as part of the mix-net integration. It is then filled by contributions from all administrators. A final service parameter configuration is agreed upon and signed by all participants in a common proof-of-agreement statement. The whole procedure is coordinated through the PANORAMIX Configuration Service.
4. Extract and install local configuration files from the service parameter configuration statement. This mechanism is also mix-net specific and must be provided as part of the mix-net integration.
5. Launch the local server software with the extracted deployment parameters.
6. Wait for software termination or any other security actions that must be taken as part of the role of the specific administrator in the service architecture (propose or consent to business logic actions).

The PANORAMIX wizard can be utilized to walk through steps 2 to 6 in a unified and user-friendly manner.

When the administrators manage to set-up their mixing servive through the PANORAMIX framework, the corresponding configuration is representated by a JSON[1] (JavaScript Object Notation) document within the system. As an example, consider the case of the Sphinx decryption mix-net [5] has been integrated in PANORAMIX (as you can recall from Deliverable 5.2,). A corresponding configuration, will be represented by the values below:

```
static_sphinxmix_spec = {
     '.negotiation.static_sphinxmix': {},
     'content': {
          'mixnet_name': {},
          'messaging_service_url': {},
          'routing': 'static',
          'sphinx_params': {
               'group': {},
```

[1]https://www.json.org/

```
                    'head_len': {},
                    'body_len': {},
                },
                'cycle_mix_size': {},
                'mixers': {
                    '?': {
                        'public_key': {},
                        'input_from': {},
                    },
                },
            },
        }
```

All the values should be instantiated through the wizard by the corresponding administrator, including the mix-nets name, the URL of the messaging service, the public key of the mixer and more.

3.3 E-voting

The e-voting use case (which involves Zeus [7]) utilizes all three services of PANORAMIX. The registration service is used to introduce, authorize, and identify participants with their cryptographic credentials. The configuration service is used to produce service deployment parameters that cover two server administrator roles: trustees and mixers. We provide information regarding the two roles in the subsections below.

Once the service is configured and deployed, clients can use the PANORAMIX messaging service to submit votes irrespective of what mix-net technology is used at the back-end. The choice of the mix-net technology can vary either at the installation level (i.e. different Zeus installations using different mix-nets) or at the event level (i.e. different elections at the same service using different mix-nets).

3.3.1 Trustee Role

Each trustee launches the PANORAMIX software to

1. Generate, keep, and contribute a secret share to the election encryption key.
2. Validate the shares of other trustees.
3. Agree on basic parameters for the election including name, ballot type and content, date, voter list and mixer list.
4. Propose and consent to put the election process to the corresponding stage (creating, voting, mixing, decrypting).
5. Validate mixing and (partially) decrypt the mixed ballots.

3.3.2 Mixer Role

Each mixer utilizes the PANORAMIX software to:

1. Agree on the mixing algorithm and the security parameters.
2. Receive encrypted votes and shuffle them providing a proof of correctness.

3. Validate other mixer's shuffles (optional).

Specific service installations might require additional roles that are tied to mix-net cryptography or general work-flow. They will have to implement those additional roles (e.g. auditor) by providing an integration as described in Section 4.5.

3.4 E-mail

The e-mail use case (built on the Katzenpost system developed in WP7) involves the configuration service and the messaging service. A Katzenpost network is assumed to be independently operational. The framework is then employed to help server administrators easily launch the Katzenpost software with the correct configuration to join the Katzenpost network, while giving them a comprehensive view of the network parameters to review and consent to.

Example parameters include the cryptographic algorithms and standards used (for instance x25519 certificates), the identity of the network authority, the type of the network authority (e.g., voting or non-voting), the algorithm for encryption key derivation, the era duration for encryption key derivation, and overlap time. The registration service is not used as the identity of nodes in Katzenpost is part of its core functionality.

3.5 Anonymized Surveys and Statistics

In WP6 we have demonstrated the use and benefits of the privacy-enhancing techniques that the PANORAMIX framework can provide in anonymized surveys and statistics. In particular, as described in Deliverable 6.2, a specific client adapter (the use case software depicted in Figure 3.1) covers differentially private anonymization of client data by sending them to a corresponding mix-net through PANORAMIX.

Example parameters include the number of mix-net nodes, the number of messages need to be gathered before mixing is started (i.e., the "batch size" parameter which influences performance and privacy protection), the cryptography used and more.

3.6 Messaging via MCMix

The MCMix system developed in WP3 comprises of the following entities: **users**, the **entry server** and the **MPC servers**. Initially, users register to the entry server submitting their public-keys and they also exchange private keys with the MPC servers. The main role of the MPC serves is to execute Multi-Party Computation (MPC) protocols, which is the main ingredient of the MCMix system. The role of the entry server is to gather messages sent by clients and to forward them synchronously to the MPC servers. In addition, the output of the MPC servers is delivered to the entry server, and clients contact it periodically to check if there are incoming messages. In order for privacy to be preserved, the communication between a client and an MPC server is end-to-end encrypted, using the private key exchanged during the registration phase.

The MCMix system comprises of two main protocols, namely the *dialing* and the *conversation* protocol. In order for a party, say A, to initiate a conversation with another party, say B, A needs to participate on the dialing protocol, at the end of which B becomes aware of A's intention to initiate a conversation. After this step both A and B participate in the conversation protocol, that enables them to exchange messages. One can think of the MCMix system as a voice-telephone system with text in place of audio, in which every user can be in conversation with at most one person at a time. The two protocols depend on client-server MPC, which is executed by the MPC servers, and allows users to exchange messages; *auxiliary information* capturing the actions "A calls B" or "A sends message m to B", is also included in the messages. In particular, in the dialing protocol, the auxiliary information is a users username, while in conversation it is a "dead drop": a value shared between two users who wish to converse, but that no other user can calculate. For both protocols, the MPC servers implement a sorting algorithm, which is performed (obliviously) so that messages having the same auxiliary information are placed next to each other. Also, in both protocols users submit encrypted payloads (messages) to the entry server, i.e., usernames in the dialling protocol as well as "dead drops" in the conversation protocol are all encrypted and private. The entry server modifies the payloads by appending wire IDs associated with the users, which allows the output payloads returned by the MPC servers to be delivered to the correct users. Privacy against the entry server is ensured as the payloads are encrypted before being passed to the entry server. Also, the MPC servers only learn secret-shares of the input to the computation, thus privacy is preserved assuming an honest majority. After the MPC protocol has run, the payloads are returned to the users via the entry server. This server manipulates the payloads by removing the wire IDs and then makes them available to be requested by clients.

The MPC servers of the MCMix system back-end have been implemented using the Sharemind system [2]. This had many advantages as from the beginning of the project, Cybernetica, the company that develops Sharemind, provided a Debian virtual machine preloaded with the MPC server binaries and SecreC[3] compiler, enabling a fast and convenient project initiation.

The main requirements for the client-facing components of this project are simple: securely manage connections with clients, accept and return data for clients, and initiate and manage rounds of the MCMix protocols. Therefore, choosing a language and framework that allowed for fast development was critical. The components also had to provide the required security properties and ease of future extension. These constraints and requirements led to a choice of writing the front end in the Python language.

Clients have been implemented for Android, which is a is a Linux based operating system with a suite of tools freely available for developing applications. We chose to develop the client program for Android Version 7 (Nougat) and above as this was the most recent major release when initiating the project. However, at all opportunities we chose classes and library options that will allow future developers to easily port the application back to previous versions if required. We wrote the application in Java (Version 7) as this is the standard language for Android development. PANORAMIX framework integration is intended to be of the similar type as the Katzenpost system.

[2]See `https://sharemind.cyber.ee/`.

[3]The programming language supported by Sharemind.

4. Architecture and Software

We provide a description of the architecture of the PANORAMIX software toolkit. Specifically, we describe the software that implements the services that we described in Chapter 2. Figure 4.1 illustrates the architecture of the final system and involves the *controller* system, and either *contributor* entities or *client* computers.

4.1 Components

Every registered contributor entity contains a *wizard* component. This component can be used by the administrator to set up the mix server, which then will act as an authenticated mix-net peer. The mix server contains two basic components, that is the *crypto module* and the PANORAMIX *client*. In particular, the PANORAMIX client initializes the crypto module after interacting with the controller through the corresponding endpoints. Each user computer contains a local agent with the same components together with the corresponding application (e.g. related to e-voting).

4.2 Registration

In subsection 2.2 we provided an overview of the registration service. Here, we provide further information regarding the corresponding module as part of the architecture and in more detail.

Figure 4.2 illustrates how the registration is done. We observe that registration is managed by the PANORAMIX controller. On the other hand we have different peers which can be mix-net contributors, administrators, users of the service provided (e.g. voters), and more. Initially, peers send their key to the service so that the service can identify them. Then, the service provides a corresponding certificate that will in turn predetermine the role(s). Recall that a peer can have more than one role in the context of a use case, i.e. in the e-voting use case a trustee can be also a mix-net contributor.

The registration component comes with a an API. In the following, we provide the basic methods:

- GET_INFO: returns a globally unique identifier for this instance of Registration API.
- REGISTER_PEER: with the following parameters: `common_name` and `signed_public_key`.
- RESPONSE: returns the `peer_certificate`.
- MAKE_ATTRIBUTION_REGUEST: with the following arguments: `issuer_common_name`, `target_common_name`, `attributes`, `issuer_signature`.
- LOOKUP_PEER_REQUEST: arguments can include either the `common_name`, the `public_key` or other `attributes`.

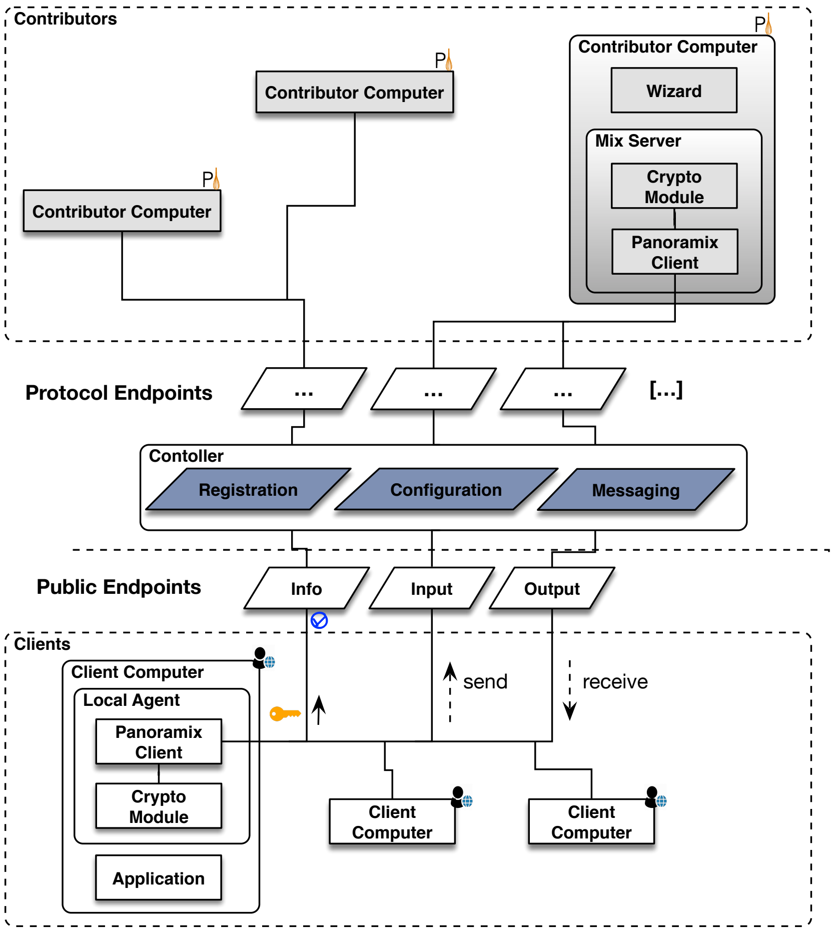

Figure 4.1: The PANORAMIX Software Architecture.

- RESPONSE: provides a list of `peer_certificates`.

4.3 Messaging

Applications either send or receive messages through a mix-net via a software component which we call *Local Agent*. This is provided by the system as a standalone software service and corresponding client software library and command line tools that takes care of all the cryptographic details as well as the PANORAMIX internals, i.e. it encrypts and decrypts messages and manages any cryptographic keys needed.

The API methods of the message are the following:

- GET_INFO: this method has the same functionality as the corresponding method of the registration module.
- CREATE/UPDATE_INBOX_REQUEST: with the corresponding arguments: `peer_owner_common_name`, `inbox_ACLs`, `peer_owner_request_signature`.

The PANORAMIX inbox has three specific and simple API methods:

1. SUBMIT_MESSAGE_REQUEST with the corresponding text as an argument,
2. READ_MESSAGE: with the message indexes to read as arguments,

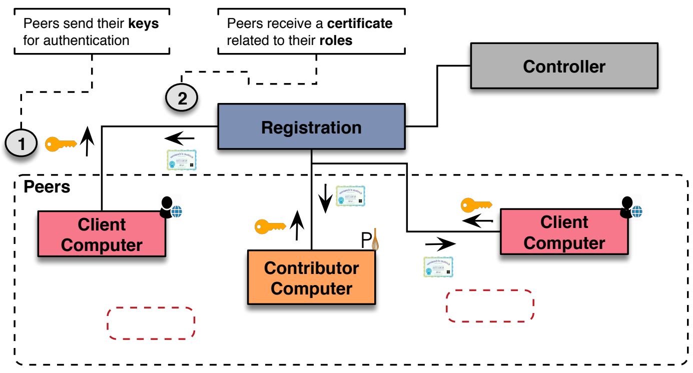
Peers send their keys for authentication
Peers receive a certificate related to their roles
Controller
1
2
Registration
Peers
Client Computer
Contributor Computer
Client Computer

Figure 4.2: The registration module. Peers send their keyes and recieve corresponding certicitates regarding their roles.

3. EXTRACT MESSAGE: which can read and remove messages from an inbox.

4.4 Configuration & Setup

To make the process of setting up a mix-net an easy task, we have developed an interactive wizard that guides contributors through the parameter selection, allowing them to confirm or counter-propose different values. This wizard can be tuned case-by-case to interactively process only a subset of the actual parameters, so that it does not become too difficult to use.

Figure 4.3 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 and launch the software. Finally, there are actions that are use-case related. Hence, the final step could take place multiple times.

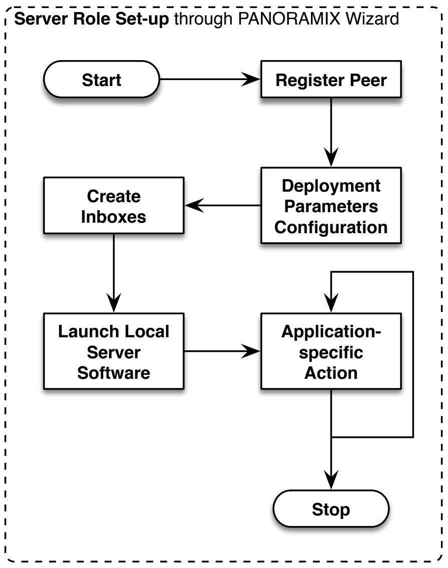
Server Role Set-up through PANORAMIX Wizard
Start
Register Peer
Deployment Parameters Configuration
Create Inboxes
Launch Local Server Software
Application-specific Action
Stop

Figure 4.3: The workflow behind the set-up of a server role through the PANORAMIX wizard.

The API methods used by the configuration service are:

- GET_INFO: this method has the same functionality as the corresponding registration method.

- CREATE_NEGOTIATION_REQUEST: an optional parameter; could be the `negotiation_id`.
- CONTRIBUTE_REQUEST: the following arguments are required: `peer_common_name`, `contribution_text` and `peer_request_signature`.
- GET_CONSENSUS.
- RESPONSE: returns that consensus text (if a consensus reached) and a list of contributor peer signatures.

4.5 Mix-net Integration Interface

In order to integrate a mix-net into the PANORAMIX framework, the integrator engineer must implement an interface to connect the mix-net software with the PANORAMIX framework. The interface includes:

a. An adapter implementing the generic message submission API of PANORAMIX, that will translate PANORAMIX message submissions into native mix-net messages.

b. A service parameter configuration template document with provisions for every server administrator role that will be supported (e.g. trustee, mixer).

c. A script that will extract local server configuration files from the common service parameter configuration document, for each different role.

d. A script that will apply the extracted configuration files into the local system so that they will be available and loaded by the mix-net software.

e. A script that will launch the actual local server software for each role.

5. Validation

To validate the PANORAMIX framework we have integrated two more mix-nets (apart from the ones that we have discussed in other deliverables such as 5.2 and 5.3) and developed a test suite to examine its functionality.

5.1 Integrated Mix-nets

One of the basic requirements of the PANORAMIX system is to support different mix-nets. Currently, our system supports a number of different mix-nets, including: (a) the Verificatum Mix-Net (VMN) [2], (b) a prototype mix-net based on the re-encryption mix-net designed by the PANORAMIX team at U. Tartu, Fauzi et al. [6] (also known as hat-shuffle). In this deliverable part we describe the integration of these two and the corresponding benchmarking we performed after the integration to observe their efficiency. Note that these are the latest mix-nets that were integrated and this is why we discuss them here. Recall that we have also ported a number of other mix-nets to PANORAMIX framework including: Sphinx [5] (as discussed in D5.2) and the Sako-Kilian re-encryption mix-net [4], which in turn is used by the Zeus [7] e-voting application (as discussed in D5.3).

5.1.1 Hat-shuffle Integration

module	**phase**	**10,000**	**10,000-P**	**100,000**	**100,000-P**	**200,000**	**200,000-P**	**1,000,000****
CRS	Generation	3.6220s	2.2555s	27.1784s	10.8926s	49.5121s	19.4441s	218.1849s
	Serialization	1.7291s	—	15.1980s	—	29.1654s	—	152.4044s
	Total	5.3511s	—	42.3764s	—	78.6775s	—	370.5893s
encrypt	CRS Deserialization	0.4397s	—	3.3140s	—	6.2664s	—	31.2038s
	Messages Deserialization	0.0946s	—	0.5895s	—	1.0446s	—	4.0429s
	Encryption	27.0073s	—	270.0997s	—	542.9835s	—	2700.1721s
	Ciphertexts Serialization	0.4681s	—	4.6864s	—	9.4914s	—	50.7118s
	Total	28.0097s	—	328.6896s	—	559.7859s	—	2786.1306s
prover	CRS Deserialization	1.0799s	—	8.5572s	—	16.3996s	—	56.2043s**
	Ciphertexts Deserialization	0.7445s	—	7.2519s	—	14.2327s	—	45.9318s**
	Prove	13.3515s	9.2914s	112.8631s	69.9510s	220.3691s	138.3695s	389.7539s**
	Proofs Serialization	2.7578s	—	22.1532s	—	52.1241s	—	101.1021s**
	Total	17.9337s	—	150.8254s	—	303.1255s	—	603.2646s**
verifier	CRS Deserialization	0.9295s	—	8.5857s	—	17.2045s	—	58.4569s**
	Ciphertexts Deserialization	0.6768s	—	6.8215s	—	14.3577s	—	42.6796s**
	Proofs Deserialization	1.4613s	—	14.9167s	—	30.0238s	—	103.6393s**
	Verify	26.4480s	8.2774s	248.5492s	74.1561s	494.0374s	150.2706s	1655.6244s**
	Total	29.5156s	—	278.8731s	—	555.6234s	—	1860.4001s**
decrypt	Messages Deserialization	0.0895s	—	0.5400s	—	1.1618s	—	2.0494s**
	Ciphertexts Deserialization	0.5507s	—	5.7847s	—	11.5266s	—	36.8059s**
	Table	9.4963s	—	93.6353s	—	184.4576s	—	408.1059s**
	Decryption	9.0543s	—	91.3663s	—	190.1704s	—	381.1743s**
	Messages Serialization	0.0144s	—	0.1591s	—	0.2904s	—	0.5766s**
	Total	19.2052s	—	191.4854s	—	387.6068s	—	828.7121s**
mix *		0:47m	0:25m	7:10m	3:35m	14:19m	7:18m	41:03m**

Table 5.1: Hut-shuffle efficiency benchmarks.

* 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)

Hat-shuffle was designed in the context of PANORAMIX [1] In particular, it is a non-interactive zero-knowledge proof (NIZK) [3] shuffle argument. We have discussed about the initial integration in the context of the e-voting use case in Deliverable 5.3. Since then, there were several software updates and here we provide evaluation of the mix-net as part of the PANORAMIX framework.

One of the main components of this mix-net involves a common reference string (CRS). Specifically, the CRS model incorporates the assumption that a trusted setup in which all involved entities get access to the same string CRS taken from some distribution D exists. Schemes proven secure in the CRS model are secure given that the setup is performed correctly.

We have performed extensive tests regarding the performance of hat-shuffle integration. In particular, 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 encryption, proving, verification and decryption. Table 5.1 shows the results of the tests. Our benchmarks included a different number of messages each time. We started with 10K messages, continued with 100K, then 200K and finally 1M messages. Note that we also did tests where threads were running in parallel. Our experiments were performed in an Ubuntu machine (16.04) with 4 CPUs, 8GB RAM and 40GB system disk.

Table 5.2 presents the different sizes of the files that are generated, for the different numbers of messages. 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.

We need to highlight here that hat-shuffle handles 1M messages (in the context of the voting use case, messages are votes) efficiently. Recall that Zeus (the e-voting platform involved in

File	**1000m**	**10,000m**	**100,000m**	**200,000m**	**1,000,000m**
ciphertexts	829K	8.1M	81M	162M	809M
CRS	1.3M	13M	124M	248M	1.2GB
messages	87K	865K	8.5M	17M	84M
proofs	2.2M	21M	210M	420M	2GB

Table 5.2: Size of files for different number of messages (**m**) in hat-shuffle.

messages	initialization	mix	ciphertexts	verify
1000000	0:54	18:42	1:32	18:42
500000	0:27	8:04	0:43	8:04
100000	0:08	1:38	0:10	1:38
50000	0:06	0:53	0:06	0:53
10000	0:03	0:17	0:02	0:17
5000	0:03	0:12	0:02	0:12
1000	0:02	0:07	0:00	0:07
10000	0:04	0:43	0:02	0:43

Table 5.3: Verificatum efficiency benchmarks.

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 it is described in Deliverable 5.4.

5.1.2 Verificatum Integration

Verificatum is a well-known open-source re-encryption mix-net. It is a stand-alone server software written in Java, with complete documentation. Note that the integration of Verificatum, is an example of how a mix-net designed outside the PANORAMIX project can be easily incorporated into our toolkit—which in the big picture, demonstrates the extensibility of the framework we have developed.

A typical Verificatum workflow is that the administrators 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.

The Verificatum servers automatically create shared keys for encryption and decryption of ciphertexts, proceed with the mixing of ciphertexts, check each other's proofs, and make the final results available locally to each node administrator.

However, in order to integrate Verificatum into the more general PANORAMIX framework we had to isolate the actual mixing phase from the typical Verificatum workflow:

a. The encryption keys are not generated by Verificatum itself but are part of the initial deployment parameters agreed upon by all mixnet contributors.

b. 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 mixnets, and uses the Verificatum proof checker toolkit to link and verify the mix contributions into a proper mixnet.

messages	proofs size
1000	2.9 MB
5000	14.4 MB
10000	28.8 MB
10000-2	47.1 MB
50000	144 MB
100000	288 MB
500000	1.4 GB
1000000	3 GB

Table 5.4: Size of files for different messages in Verificatum.

c. Since encryption keys are not generated and managed by Verificatum, for the decryption of mixed ciphertexts, Verificatum is asked to skip decryption, which is handled at a higher level in a way similar to (b.) above.

We have evaluated Verificatum in a similar manner to hat-shuffle. Table 5.3 shows the efficiency of Verificatum and its functionalities for different numbers of messages. Table 5.4 displays the sizes of the files as the messages increase. Once again we observe that there is a linear correlation.

5.2 Testing

It is important to be able to prove that the integration of each mix-net was successful and that the mix is working properly at all times as the code bases change. A robust approach for doing so is to feed the PANORAMIX interface with random input and validate the output, repeating over many cycles. We have implemented this approach for random input testing by setting up an environment for the test. In this context, an integrated mix-net is initialized and is able to properly run with real input and produce also real output.

Each mix-net implementation has specific tests. The tests are being monitored regardless of the integration. Nevertheless, we have to prove that the integration of each mix-net was successful and that the mix is working at all times.

A stable method for testing is to feed the interface random inputs and validate the outputs. This is done on many cycles. We have implemented a corresponding testing environment based on this approach.

For every cycle, random inputs of ciphertexts are created by encrypting a set of plaintexts. Then, the plaintexts are recorded while the ciphertexts are provided to the mix-net. After the mix-net processes the ciphertexts, it produces a new set of shuffled ciphertexts as output. The output is then retrieved and decrypted. Finally, the shuffled plaintexts are compared to the original ones.

There are two different comparisons. First, the original and the output plaintexts are compared. We assume that the sets are different, because they original ones have been shuffled. Then, the two sets are sorted and then compared. In this case, we assert that they are identical. This is because we expect them to be exactly the same set without any corruption or omission.

The testing process is due for all mix-nets and beyond end-to-end testing and validation. It offers developers and system administrators quick insights on their configurations, integration, and performance.

6. Conclusion

The Final System is a robust, flexible implementation of a framework that effectively provides the infrastructure for the different PANORAMIX use cases with reasonable performance, clear and easy programming interfaces, and services easy to deploy.

We have addressed the feedback coming from different work packages with regards to these three aspects. We also demonstrate the functionality of the framework through the implementations of our three use cases.

Thanks to the intensive software testing of the software, the code quality has been improved to reach a production ready level, and future maintenance has become easier thanks to the automated testing suite integrated in the system.

Full functionality and improved maintainability close the gap between the integrated system delivered in D4.3 and the final system. Use cases have been able to use it to accomplish their goals, and third parties are now able to leverage the same infrastructure to provide privacy-preserving communications based on mix networks.

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] 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.

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

[5] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, SP '09, pages 269–282, Washington, DC, USA, 2009. IEEE Computer Society.

[6] Prastudy Fauzi, Helger Lipmaa, Janno Siim, and Michal Zajac. An efficient pairing-based shuffle argument. In *ASIACRYPT (2)*, volume 10625 of *Lecture Notes in Computer Science*, pages 97–127. Springer, 2017.

[7] 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. The PANORAMIX Toolkit

We provide the general usage of the PANORAMIX toolkit for administrators and developers.

GENERAL

usage: `mixtool [-h]`
{init-workspace, register, configure, deploy, launch, watch} ...

positional arguments:
{init-workspace, register, configure, deploy, launch, watch}

`sub-command`	Help
`init-workspace`	Initialize a workspace
`register`	Connect to a service and register key
`configure`	Specify and negotiate deployment parameters
`deploy`	Install agreed configuration
`launch`	Launch app with the deployed settings
`watch`	Watch service state

optional arguments:

`-h, --help`	show this help message and exit.

INIT

usage: `mixtool init-workspace [-h] [--name NAME] [--type TYPE]`

Initialize a workspace.

optional arguments:

`-h, --help`	show this help message and exit
`--name NAME`	local session identifier
`--type TYPE`	type of mix-net service

REGISTER

usage: `mixtool register [-h] [--service-url SERVICE_URL] [--username USERNAME]`
`[--key-location KEY_LOCATION]`

Connect to a PANORAMIX service and register key.

optional arguments:

`-h, --help`	show this help message and exit
`--service-url SERVICE_URL`	URL of the Panoramix service
`--username USERNAME`	Register as this user name
`--key-location KEY_LOCATION`	Location of crypto key

CONFIGURE
usage: `mixtool configure [-h] [--role ROLE] [--config-file CONFIG_FILE]`

Specify and negotiate deployment parameters.

optional arguments:

`-h, --help`	show this help message and exit
`--role ROLE`	assume this role in the service
`--config-file CONFIG_FILE`	load local params from file

DEPLOY usage: `mixtool deploy [-h] [--path PATH]`

Install agreed configuration.

optional arguments:

`-h, --help`	show this help message and exit
`--path PATH`	deployment path

LAUNCH
usage: `mixtool launch [-h]`

Launch application with the deployed settings.

optional arguments:

`-h, --help`	show this help message and exit

WATCH
usage: `mixtool watch [-h]`

Watch service state.

optional arguments:

`-h, --help`	show this help message and exit

B. License

In the original PANORAMIX proposal (section 2.2), it was stated that:

> Code of the generic API will be released under a dual-licensed open-source scheme to allow commercialization in both proprietary codebases and utilization of the code in the wider open-source community. This will require dual licensing between the GPLv3/AGPL and a commercial-friendly license such as the BSD license. This should let us pursue market opportunities with the widest possible kinds of exploitation plans while keeping a core open source infrastructure for Europe that all companies and researchers can use with confidence and equally access.

Conforming with our initial objectives, the code framework has been released under the GNU Affero General Public License (AGPL, see `https://github.com/grnet/panoramix/blob/develop/COPYING`). The choice of AGPL was dictated by the following reasons:

- We want a GPL-like license, so that changes to the code will be made publicly available under the same license. The PANORAMIX partners feel that it is particularly important in projects with high security requirements to ensure that developments and improvements are brought out in the open.
- A simple GPL (meaning, not AGPL) license, does not require the publication of code in products that are used for delivering services and in which it is not necessary to distribute the code of the product. The PANORAMIX code is such a server-side product, which means that releasing it under a GPL license would not be meaningful. Indeed, the AGPL license was evolved from GPL in response to exactly this kind of situations.
- The PANORAMIX partners can of course release the source code, after an appropriate agreement, under a more commercial-friendly license such as the BSD, to interested partners. In this way, specific commercial opportunities will be explored, without impinging on the overall open source orientation of the project.