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

Integrated System

Deliverable D4.3

January 31, 2018
PANORAMIX Project, # 653497, Horizon 2020
http://www.panoramix-project.eu

Horizon 2020
European Union funding
for Research & Innovation

Revision History

Revision	Date	Author(s)	Description
0.1	2017-11-29	RG (KUL)	Proposed table of contents, Documentation appendices
0.2	2018-1-2	DM (GRNET)	First draft of requirements
0.3	2018-1-11	DM (GRNET)	Software architecture and Validation
0.4	2018-1-11	GT (GRNET)	Elaborate on requirements per use case
0.5	2018-1-12	GT (GRNET)	Future development notes
0.6	2018-1-15	RG (KUL)	Introduction and Conclusion
0.7	2018-1-31	DM (GRNET)	End-to-end testing
1.0	2018-1-31	MW (UEDIN)	Final review and submission to the EC

Executive Summary

This deliverable describes the software package that has been used by partners to develop the latest versions of their systems, being validated by the collaboration between the corresponding teams and providing the implementation of all the requirements not addressed by D4.2.

The Final System has been designed in order to facilitate its deployment as a scalable and robust Internet service whose functionality fulfils the goals of a privacy preserving anonymous communication system ready to be used by clients in an easy and flexible way.

Future development will address any remaining issues that arise from the real world usage of the Panoramix framework, effectively integrating mix-net functionality into applications with no need for further documentation rather than the simplified one already provided.

Contents

1. Introduction

This deliverable describes the Integrated System, which implements the remaining requirements which were not addressed in the Minimum Viable Product. It constitutes a complete system that has been shown to work with the three proposed use cases: the API is stable, the Software architecture has proven to be useful for them to make use of the system functionality, and the documentation annexed explains everything needed for the System Administrators and the Software Developers to utilize the library.

By working together with the use case developers, the Integrated System has evolved towards an easy to use and fully featured product that third parties with different aims can use seamlessly. Furthermore, the latest developments in WP5 and WP7 already target this version of the library, and the communication between both teams has enabled rapid and effective iterations of the API and the helper tools.

The aim of the document is to detail how the implementation of the system fulfils all requirements needed by each use case (Chapter 2), how the software architecture implements them in a flexible and easy to use manner (Chapter 3), and what has been the validation process of the system (Chapter 4).

2. Fulfilled requirements per use case

In this section we describe the requirements that must be met in the context of PANORAMIX. We start with the general requirements that apply to all use-cases and then focus on specific requirements per use-case.

2.1 General requirements for all use-cases

There are several requirements that are common for all the use cases of the project. Here, we discuss these requirements in detail and highlight their role in relation to PANORAMIX.

2.1.1 Peer authentication and Public Key Infrastructure

All security applications and specifically mix-net infrastructures must distribute trust among different actors that are assumed to be independent.

The fundamental mechanism with which each of those actors can authenticate themselves and participate into protocols is by having 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.

The inherent public key infrastructure problem is how different actors know the public keys amongst themselves. A typical implementation is to establish a central authority that will sign certificates that maps public keys to application-specific roles and identities such as email or names. However, the central authority becomes a central security and privacy point of failure, and this has been proved in practice several times. A central authority is still supported by PANORAMIX, but we would like to add flexibility so that applications themselves may establish the actor-to-key correspondence in a way that they know and guarantee is secure to their use-case.

PANORAMIX peer authentication offers peer-to-peer discovery of public keys, and through the negotiation mechanism described later the mix network can establish consensus of the global peer list. The mechanism to do that is through application-specific contacts. The basic trusted information that human actors or trusted systems have before the establishment of a Public Key Infrastructure (PKI) provide the means to communicate privately and securely with each other. Be it an email address, or a firewall-ed VPN connection, mediation through a third party, or even a hand-to-hand exchange between administrators, trust is established by exchanging public keys through an initially available communication channel. PANORAMIX needs to facilitate this exchange and key discovery through its architecture and APIs.

2.1.2 Usable and Secure Mix-contributor Configuration and Audit-Log for Administrators

Many security-critical applications and protocols, including mix-nets, distribute trust among network actors that are considered independent and are secured in different administrative domains. Nevertheless, these distributed actors need to collaborate in order to correctly execute a protocol and the coordination between them and its security implications are often overlooked.

It is often assumed that certain consensus over technical details pre-exists. For example, what software version is run, which cryptographic algorithms are used, what tuning parameters are used to initialize software, what are the different actors, what kind of keys they possess, and which are the actor's network endpoints. This consensus is often naturally occurring among stakeholders that are strongly interested in a specific application and its mix-net. However, their strong stakeholder interest makes those actors well-known, prone to attacks, corruption and / or coercion. Ideally, PANORAMIX should offer an easy, yet trusted way for disinterested parties to be able to participate.

There are several points to consider for this:

- **Awareness of the consensus** Human or automated actors must be able to review and explicitly agree on the consensus. Having the consensus formed implicitly by central authorities undermines their independence. Even software distribution authorities can undermine independence.
- **Usability** It should not require an expert to navigate the configuration and the initial consensus. It is expected that users will not understand all parameters, but they have to be able to distinguish among different parameters and choices and be able to relay information to expert consultants if they choose to do so.
- **Auditability** When considering that disinterested parties acquire responsibilities by being an actor in a cryptographic protocol such as a mix-net, auditability becomes important.

Since the disinterested actor has, by definition, no natural knowledge of the process and application environment, they will resist responsibilities in case they are manipulated by better informed but malicious stakeholders.

This has been observed as a strong trend in the e-voting use-case experience. To counter-act this trend, PANORAMIX needs to offer:

1. A way to technically record the protocol requirements for each actor in the protocol so that their responsibilities are strictly defined.
2. An audit log of all configuration parameters and actions by the actor and their software installations. The audit log can be used by the actors themselves to review their own security and to prove to investigating authorities that they did no wrong.

This is especially important if PANORAMIX wants to encourage mix-net technology to be adopted in an environment where responsibility for user privacy is being legislated as a core component of the information technology industry For instance, the European Union General Data Protection Regulation (GDPR) [1] replaces central certification authorities with self-assessment processes. This shifts the weight of risk analysis and security responsibility to the individual organization. PANORAMIX tools are designed for increased transparency and control of technical parameters and as such will be a valuable asset for meeting the regulation requirements.

2.1.3 Integration Between a Mix-net Implementation and the PANORAMIX Controller

As discussed earlier, a unified software controller for authentication / PKI, configuration, and audit log management is an important contribution that PANORAMIX can deliver to its users.

This common view, however, requires that different mix-nets can be plugged in and be controlled through a modular interface. Hence, the PANORAMIX controller software must provide an interface so that different mix-nets can be integrated. At the very least, integration means a module that encodes configuration parameters, actor roles, and can produce configuration files. At best, integration could mean that the mix-net is controllable at runtime. The minimum functionality is included in the Integrated System, and some control features are expected for the final demonstrators.

2.2 Use-case Specific Requirements

In this section we describe the different requirements that are related to the use-cases of the project, expanding on the requirements presented in D4.1 and D4.2 to include the specific requirements met in this iteration.

2.2.1 E-voting

In the e-voting use-case we have identified the following requirements

- **Trustees must be able to control the election procedure, including mixing**. This requires a usable and intuitive user interface that will present all decisions and actions in a similar way so that the trustees can have sufficient overview themselves and not rely on delegation to more technically trained operators. Usability will also make finer control and overview possible. An important part of control is the audit log of all actions so that trustees can both be held accountable for the procedure and prove to auditors that they observed due diligence.
- **Usability and security for non-expert mix-net contributors**. Similarly to trustees, mix-net contributors should be independent yet have critical responsibilities for the process. Non-experts are hesitant to take on such a technical and important role on their own capacity. The easy solution to use contributors close to the environment of the election defeats independence. A system that is both usable and inspires safety can address this issue.

2.2.2 Messaging

Messaging in the context of PANORAMIX is related to the following requirements:

- **Mix server identity registry**. Each actor in the mix protocol is identified by a cryptographic public key. The establishment of a registry is a basic requirement.
- **Global consensus of the mix-net topology**. The role of each actor in the identity registry defines the topology of the mix-net, which is organized in layers. It is important that everyone agrees on what nodes are in what layer.

- **Requirement for periodic re-configuration**, such as adding or removing peers. In a dynamic environment, conditions change and actors come and go. There has to be a way to securely reconfigure parameters and roles at runtime.

3. Software architecture

3.1 Overview

The PANORAMIX framework assumes that each individual mix-net is organized around three (micro-)services, namely: the *registration service*, the *configuration service*, and the *messaging service*.

In particular, the PANORAMIX framework has been designed as a software toolkit that provides servers for deploying the aforementioned services, wizards to configure them, and client software to interact with them.

In the following sections we describe the functionality of each service, and present the architecture of the PANORAMIX toolkit.

3.2 Basic Services

In this section we describe the three basic services of the PANORAMIX toolkit. Figure 3.1 illustrates the different actors, clients or contributors, that connect to PANORAMIX, and PANORAMIX's different services. Recall that a *client* wants to use a PANORAMIX mix-net, while a *contributor* offers its services as a mixing node. All services are provided by the core component of the PANORAMIX toolkit which is the *controller* software. Note that each actor runs the PANORAMIX client software and software related to the mix-net being used. We further describe this software instances in Section 3.3.

3.2.1 Registration Service

The registration service fulfils the general requirement for peer authentication and public key infrastructure that offers identity management (see Subsection 2.1.1).

All actors in the mix-net must create their identity in the form of a public-private key pair and register it along with a mix-net-specific role designation. The registration service is then responsible to decide on and distribute a consistent view of the peers and roles 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 corresponding service (presented in the next section). A common pattern is to elect or assign specific identities to be authorities that can then assign or revoke roles.

Figure 3.2 illustrates how the different actors are connected to the registration service of PANORAMIX. A general use-case of the service could be the following: first a client or a contributor must create a key pair and then send it to the service. Then the service registers a role for the actor and can also authenticate them.

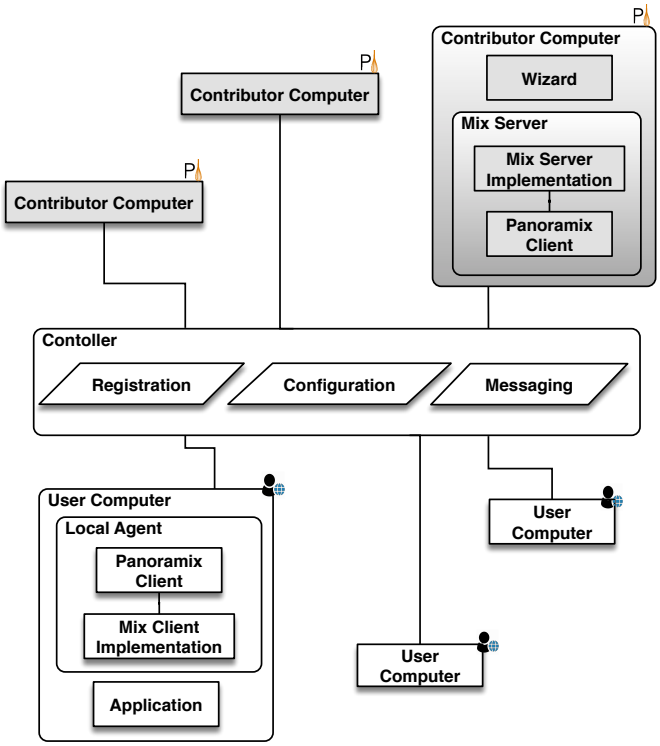

Figure 3.1: The PANORAMIX Services Architecture.

3.2.2 Configuration Service

The configuration service answers the general requirement for secure configuration and auditability by system administrators (see Section 2.1.3).

Different actors in a mix-net must independently agree on the initial setup or the current effective parameters. The agreement and configuration availability is critical for the function and security of the mix-net. An audit record of the agreements is also important for the auditability and accountability of the operation of the mix-net.

The configuration service accepts different proposals for an agreement under an arbitrary title. Then, actors negotiate through rounds where they collect the proposals of others and submit their own. Finally, the service publishes the agreements in a secure and immutable manner.

Note that mix-net implementations may perform multiple negotiations under protocol-specific labels either for initial configuration or for altering application parameters, or peer roles.

3.2.3 Messaging Service

After a mix-net has been set up and is functional, the messaging service accepts messages for delivery through the mix-net using a well known interface that should be followed by all implementations. The service also offers a message inbox where incoming messages are queued for consumption. Messages are typically sent and received by end-users. However, the messaging service may also be leveraged for internal communication among the mix-net actors.

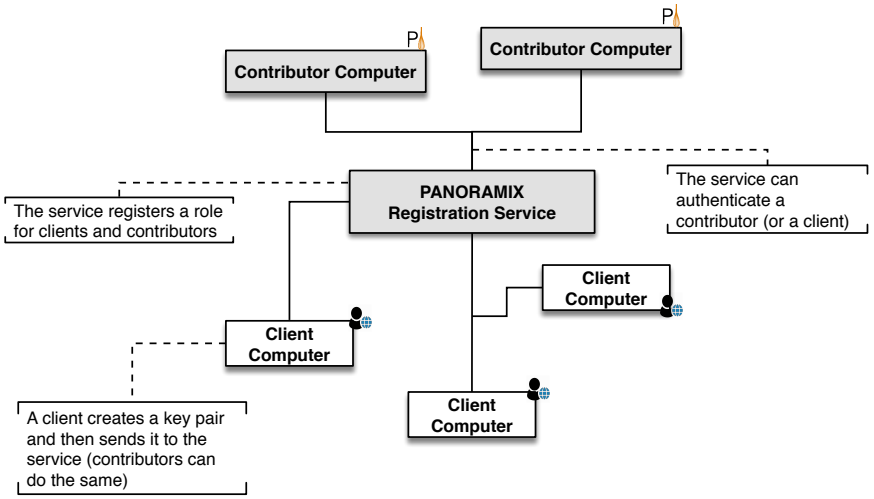

Figure 3.2: The registration service is responsible for the PKI, the authentication of the different actors and role assignment.

3.3 Architecture

We provide an overview of the architecture of the PANORAMIX software toolkit. In particular, we describe the software that implements the services that we described earlier. Note that the different mix-net implementations may only use parts of the toolkit according to their specified needs.

The architecture that we discuss here is an iteration of the initial architecture that we have already described in Deliverable 4.2. In the context of the PANORAMIX platform, there are three main entities namely: the *controller* system, the *contributor system*, and the *client computer*. The controller implements the registration, configuration, and messaging services.

The messaging service employs two kinds of endpoints (see Figure 3.3) that can be utilized by a registered client computer that will send and receive encrypted messages, and an endpoint that 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., re-encryption mix-net). Documentation regarding the endpoints and their use can be found in Appendix B.

3.3.1 Basic Components

Every registered contributor computer contains a *wizard* component. This component can be used by the administrator to set up the mix server which in turn, will act as an authenticated mix-net peer. The mix server contains two basic components, the *crypto module* and the PANORAMIX *client*. The latter initializes the former after interacting with the controller through the corresponding endpoints. Each user computer contains a local agent with the same components.

3.3.2 Sending and Receiving Messages

Applications send or receive messages through a mix-net via a software component called the PANORAMIX Local Agent that we have extensively described in Deliverable 4.2. In brief, the

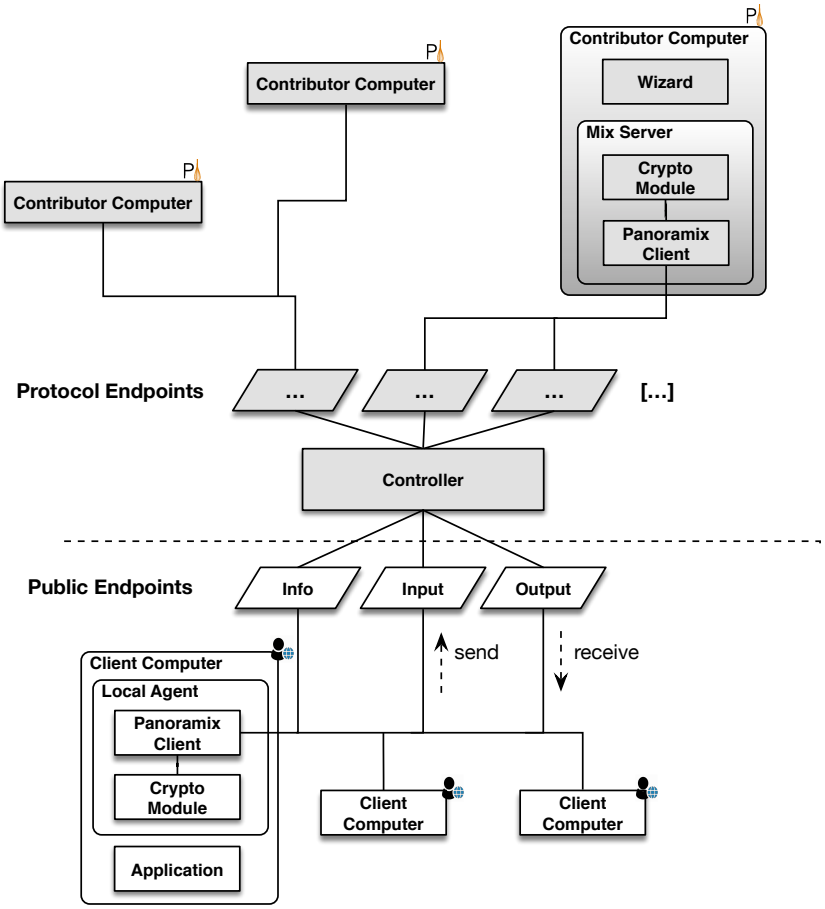

Figure 3.3: The PANORAMIX Software Architecture.

PANORAMIX local agent runs as a standalone software service that encrypts and decrypts messages, and manages any cryptographic keys needed.

3.3.3 Mix-net Setup

The mix server is employed by the different contributors to the mix. In brief, the contributors are controlled by independent parties because the mix-net relies on each of them separately to safeguard the privacy of the messages even in the presence of dishonest others. The person or the organization that wants to contribute to a mix-net must configure the mix-net server to work with the mix-net initiated by the controller. This configuration involves setting cryptographic and other parameters for the mix-net, including who the other mixers might be, and where should the messages go after the mix-net.

To aid with the process of setting up a mix-net, we have developed an interactive wizard that guides the contributor through the parameters allowing them to confirm or counter-propose different values. This wizard can be tuned case-by-case to process only a subset of the actual parameters interactively. The wizard itself will automatically agree on the other parameters as long as they match its defaults.

After initialization, the mix server enters a loop awaiting messages to process according to the parameters agreed by all. The actual details of the operation differ according to the mix-net type and specific implementation and is encapsulated within the cryptographic module.

3.3.4 A mix-net Setup Scenario

In this section we provide a simple mix-net setup scenario (similar to the one described in the Deliverable 4.2) through a sequence of GET and POST methods that take place between the different PANORAMIX entities.

To set up a mix-net, the registered coordinator must employ the PANORAMIX controller software. First, the coordinator sets up the controller database and corresponding service on a specified URL using the dedicated wizard. Then the coordinator utilizes the contributor wizard to set up the mix-net along with all the other contributors. The coordinator selects the cryptographic settings, and registers (by providing a specific identifier—in this case the identifier is 667f36f2d13a4bdc9bf518b) their personal cryptographic key to the controller.

```
"POST /panoramix/peers/ HTTP/1.1" 201 490
"GET /panoramix/peers/667f36f2d13a4bdc9bf518b/ HTTP/1.1" 200 490
```

The coordinator then selects the attributes of the mix-net and initiates a negotiation to create the mix-net in agreement with other contributors. The coordinator invites them to join the process, by sharing the controller URL and a cryptographically secure invitation. Using the same wizard, the various invited contributors can choose to approve the coordinator's proposal and finalize the creation of the mix-net. This is done via rounds of negotiations until all involved contributors agree upon the same proposal.

```
"POST /panoramix/contributions/ HTTP/1.1" 201 878
"GET /panoramix/negotiations/N7IjtRZic-RqBE-taxeXT0/ HTTP/1.1" 200 1059
"POST /panoramix/peers/ HTTP/1.1" 201 490
"GET /panoramix/peers/6130ca103bf1c4b/ HTTP/1.1" 200 490
```

The mix-net is now set up. Hence, we can provide to the end-users with the mix-net URL, which in turn, can be used to access the mix-net.

```
https://<controller_url>/panoramix/peers/6130ca103bf1c4b/
```

Nevertheless, the mix-net is not ready to accept incoming messages, before all contributors agree to create and open an inbox. This is also facilitated by the wizard. The coordinator selects the inbox attributes and then it is up to the other contributors to accept them through a round of negotiations.

```
"GET /panoramix/contributions/?negotiation=RkF20im6L2neqol HTTP/1.1" 200 1661
"POST /panoramix/contributions/ HTTP/1.1" 201 879

"GET /panoramix/negotiations/RkF20im6L2neqol/ HTTP/1.1" 200 1376
"POST /panoramix/endpoints/ HTTP/1.1" 201 447
"GET /panoramix/endpoints/ep_1/ HTTP/1.1" 200 447
```

The mix-net is now ready to accept messages on the created inbox. The contributors can use the wizard to automate the processing of the inbox. The wizard polls the inbox and checks whether it is ready to be processed. Then it calls the contributors to agree on its closure.

```
"POST /panoramix/contributions/ HTTP/1.1" 201 864
"GET /panoramix/negotiations/Bwmvzn7EmiaaHFXyq25Wy/ HTTP/1.1" 200 1361
"PATCH /panoramix/endpoints/ep_1/ HTTP/1.1" 200 654
```

Then, each contributor retrieves the messages, processes them locally using the operation specified by the endpoint and uploads the processed messages.

```
"GET /panoramix/messages/?box=ACCEPTED&endpoint_id=ep_1 HTTP/1.1" 200 2037
```

```
"POST /panoramix/messages/ HTTP/1.1" 201 2034
```

Depending on the use-case, messages may be forwarded to the contributors' endpoints for further processing, until the final results reach the outbox. Further technical details are discussed in Appendix A.

3.4 Future development

The existing software functionality is expected to have a few more development iterations to address maintenance issues (bug fixing, code cleanup).

Regarding new functionality, the e-voting and messaging use-cases are going to provide detailed configuration options for their scenarios so that example demonstrators can be developed.

4. Validation of the functionality

There are different aspects related to the validation of the PANORAMIX toolkit. One of the main aspects that has been successfully validated is the ability to port different mix-nets into the system. Another aspect involves different demonstrators. So far we have developed two demonstrators in the context of the project. We discuss these advances in the upcoming subsections.

4.1 Integrated Mix-nets

As we have discussed in Deliverable 4.2, PANORAMIX provides to interested third parties a way to integrate the functionality it provides through an easy to use and fully documented API. By using this API different mix-nets can be integrated to the platform in an easy manner. To validate the functionality of the PANORAMIX toolkit we attempted to integrate different mix-nets. We have managed to successfully integrate: (a) the Sphinx decryption mix-net [3], and (b) a Sako-Kilian re-encryption mix-net [2] (which is also used by the Zeus [4] e-voting application).

In the case of re-encryption mix-net, the trustees must specify the exact sequence on which the mixing and decrypting endpoints will operate. PANORAMIX enables the contributors to agree on these settings in a cryptographically secure way, before launching the mix-net. Through a negotiation they can formally agree on the creation of the involved endpoints (such endpoints are described in detail in Deliverable 5.2).

To support this setup, the PANORAMIX toolkit allows an endpoint description to specify the endpoints it expects to retrieve its input from. This facilitates a workflow which can be easily streamlined, with each endpoint polling other endpoints for their output. It also allows a clean separation between the mix-net's public and private interfaces (recall Figure 3.3): the ballot box's inbox and outbox constitutes the public interface; while actual processing is delegated to the other endpoints, which can be private.

In each integration we need to create a *joint peer* (see Subsection 3.3.4). This fixes the cryptographic settings across a mix-net, but does not involve any application settings. A simple negotiation is enough. In the case of the Zeus mix-net we start off a negotiation based on the following JSON (JavaScript Object Notation) canonical form:

```
[
  {
      "peer_id": "joint_peer",
      "endpoint_id": "ballotbox_election",
      "endpoint_type": "ZEUS_BALLOT_BOX",
      "links": [{"from_endpoint_id": "combine",
                 "from_box": "OUTBOX",
                 "to_box": "PROCESSBOX"}]
  },
  {
```

```
        "peer_id": "peer1",
        "endpoint_id": "mix_peer1",
        "endpoint_type": "ZEUS_SK_MIX",
        "links": [{"from_endpoint_id": "ballotbox_election",
                   "from_box": "ACCEPTED",
                   "to_box": "INBOX"}]
    },
    {
        "peer_id": "peer2",
        "endpoint_id": "mix_peer2",
        "endpoint_type": "ZEUS_SK_MIX",
        "links": [{"from_endpoint_id": "mix_peer1",
                   "from_box": "OUTBOX",
                   "to_box": "INBOX"}]
    },
    {
        "peer_id": "peer1",
        "endpoint_id": "decr_peer1",
        "endpoint_type": "ZEUS_SK_PARTIAL_DECRYPT",
        "links": [{"from_endpoint_id": "mix_peer2",
                   "from_box": "OUTBOX",
                   "to_box": "INBOX"}]
    },
    {
        "peer_id": "peer2",
        "endpoint_id": "decr_peer2",
        "endpoint_type": "ZEUS_SK_PARTIAL_DECRYPT",
        "links": [{"from_endpoint_id": "mix_peer2",
                   "from_box": "OUTBOX",
                   "to_box": "INBOX"}]
    },
    {
        "peer_id": "joint_peer",
        "endpoint_id": "combine",
        "endpoint_type": "ZEUS_SK_COMBINE",
        "links": [{"from_endpoint_id": "decr_peer1",
                   "from_box": "OUTBOX",
                   "to_box": "INBOX"},
                  {"from_endpoint_id": "decr_peer2",
                   "from_box": "OUTBOX",
                   "to_box": "INBOX"}]
    }
  ]
```

This describes a graph of endpoints where each list element is a prescription to create an endpoint. The attribute `"links"` describes where each endpoint takes its input from (be it the input of the INBOX or the PROCESSBOX). We observe that there are different types of endpoints e.g. an endpoint used only for mixing, another for partial decryption etc. The mix-net contributors (peers) inspect the definition, negotiate in order to change e.g. the endpoint_ids (or any other attributes not shown in the above definition), and finally create the endpoints according to the definition.

To start a negotiation in the case of Sphinx mix-net (static routing) we show the corresponding definition:

```
[
  {
```

```
      "peer_id": "joint_peer",
      "endpoint_id": "our_sphinx_mix-net",
      "endpoint_type": "SPHINXMIX_STATIC_GW",
      "size_min": 3,
      "links": [{"from_endpoint_id": "peer2_mix",
                 "from_box": "OUTBOX",
                 "to_box": "PROCESSBOX"}]
  },
  {
      "peer_id": "peer1",
      "endpoint_id": "peer1_mix",
      "endpoint_type": "SPHINXMIX_STATIC",
      "size_min": 3,
      "links": [{"from_endpoint_id": "our_sphinx_mix-net",
                 "from_box": "ACCEPTED",
                 "to_box": "INBOX"}]
  },
  {
      "peer_id": "peer2",
      "endpoint_id": "peer2_mix",
      "endpoint_type": "SPHINXMIX_STATIC",
      "size_min": 3,
      "links": [{"from_endpoint_id": "peer1_mix",
                 "from_box": "OUTBOX",
                 "to_box": "INBOX"}]
  }
]
```

4.2 Testing End-to-End

Each mix-net implementation has its own tests. Such tests are being monitored regardless of the integration. However, 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. After all, it is the integrated mix-net that will serve applications by receiving and mixing ciphertexts.

A robust approach for testing an application is to feed the interface random input and validate the output, repeating over many cycles. We have implemented this approach for random input testing by setting up a controlled environment for the test. In this environment, an integrated mix-net is initialized and is able to properly run with real input and produce also real output.

For every cycle, a random input of ciphertexts is created by encrypting a set of plaintexts. 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 collected and decrypted. Finally, the final shuffled plaintexts are compared to the original ones.

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

The testing process is due for all integrated mix-nets and beyond end-to-end testing and validation. It offers developers quick insights on their mix-net configuration, integration design, and performance.

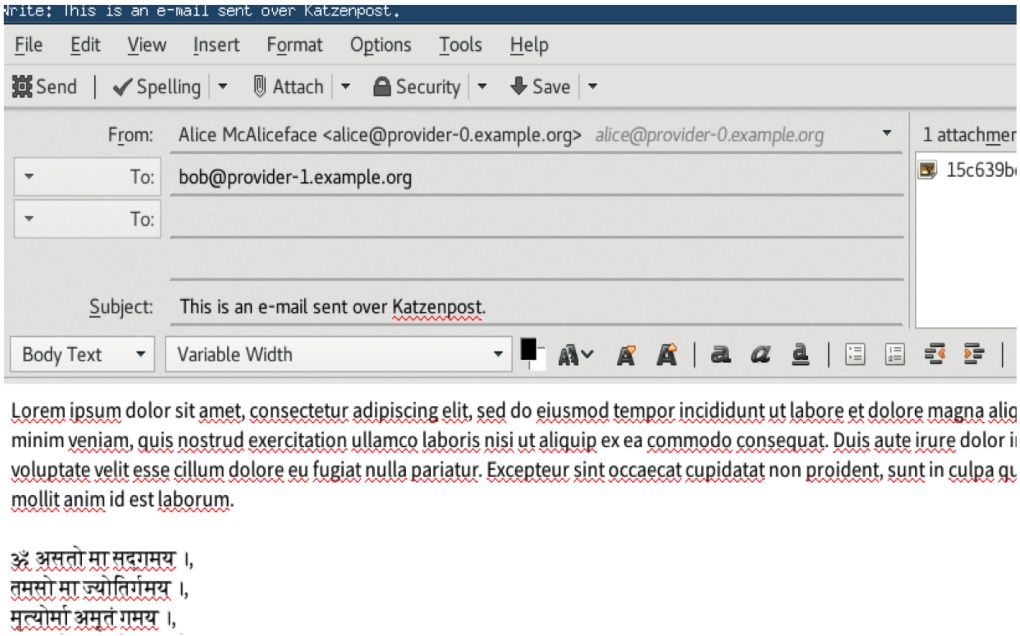

Figure 4.1: The PANORAMIX-enabled version of Thunderbird.

4.3 Demonstrators

Both of the above integrations led to two demonstrators: (1) a prototype private chat room, and (2) a Zeus version based on PANORAMIX.

The private chat room was presented as a demo at the 10th International Conference on Computers, Privacy & Data Protection (CPDP 2016), in January 2017 in Brussels. This prototype demonstrates how messages can be broadcast anonymously, without allowing an eavesdropper or even legitimate users to figure out which user sent which message (for more information refer to Deliverable 4.2).

Apart from the open-source mobile client for secure messaging discussed in Deliverable 7.2, additional demonstrators have also been developed in the context of work package 7, including a PANORAMIX-enabled version of Thunderbird[1] (see Figure 4.1).

A PANORAMIX-based version of Zeus has already been developed as we have discussed in Deliverable 5.2. We have not used this version in a real election yet. However, we are planning to do so in the upcoming months.

4.4 Continous Integration Services

Our validation also utilizes Travis CI (Continuous Integration), a hosted, distributed continuous integration service used to build and test software projects hosted at GitHub. With Travis we can specify our building and testing environment, the dependencies which must be installed before the software can be built and tested and more. When Travis has been activated for a given repository, GitHub will notify it whenever new commits are pushed to that repository or a pull request is submitted. Figure 4.2 for instance, illustrates the various tests related to the Katzenpost system (described in detail in Deliverable 7.2).

[1] `https://www.mozilla.org/en-US/thunderbird/`

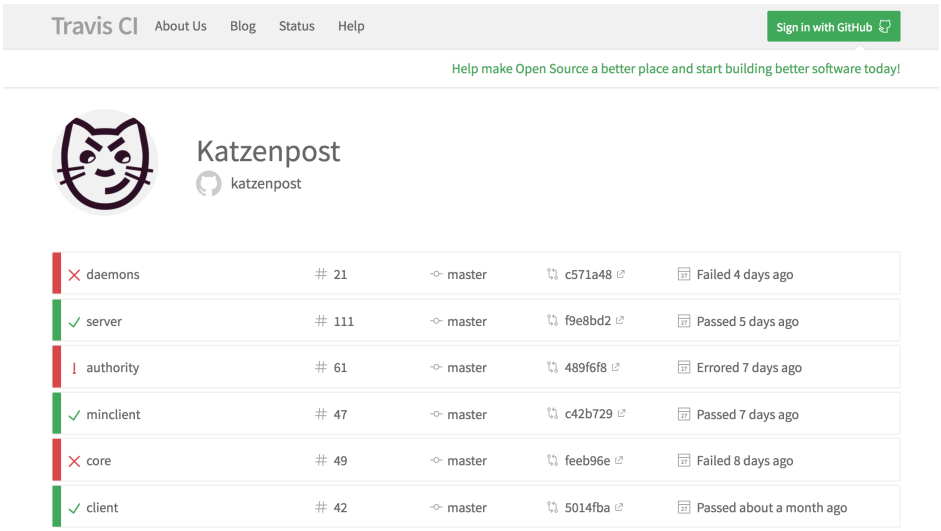

Figure 4.2: Travis CI tests for Katzenpost.

4.5 Future Scenario for Messaging Integration

In this section we discuss an initial scenario that we use as a basis to validate messaging with PANORAMIX.

The scenario follows the user experience of an independent administrator of a server. The administrator discovers a messaging mix network and joins it with their server as a new mix node.

For the first step, the administrator launches the panoramix wizard and provides the mix-net coordinator URL. The wizard connects to the coordinator API and retrieves and displays information about the mixnet and the node peers it consists of.

The administrator then requests to join the mix network. The wizard asks if there is an existing identity key to be registered with the coordinator, or if it is to create a new one. After the registration of the server as a node peer, the administrator is presented with a form listing configuration options regarding cryptographic, topology, and functional parameters, as well as other important metadata, such as terms of service, or mix-net policies and disclaimers.

After the administrator configures and accepts the form, the other mix-net nodes process and accept the join request. If there is consensus about the joining, the configuration options submitted with the form are signed-off by everyone and are returned to the wizard.

The wizard then allows the administrator to automatically translate the agreed configuration options to configuration files and then launch the mix-net software.

The signed configuration options are exported so that the administrator can inspect them for their own use or to show auditors.

5. Conclusion

The Integrated System has already been used internally to develop the latest versions of the use cases, giving solid and real world feedback that the final version of D4.3 has been able to incorporate.

The system has addressed all the requirements, the team has validated the implementation along with the interested partners and the provided documentation has been proven useful to them.

Thanks to the service oriented architecture, the development of the Final System will benefit from a well thought and state of the art Software Architecture that facilitates the deployment of the system as an scalable and robust Internet service as well as the independent development of any remaining integration issues in a particular component.

A. Documentation for System Administrators

This section provides all the information that a coordinator and a contributor will need to set up the framework. The scenario follows the steps described in Section 3.3.4.

A.1 Coordinator

To set up a mix-net, the coordinator must use the controller. Specifically, the coordinator first sets up the controller database and service on a specified URL, using a dedicated server configuration wizard.

```
% panoramix-server-wizard
welcome to panoramix server wizard!
configuration file is: /tmp/panserver
set panoramix_config environment variable to override
set catalog_url: (enter for default 'http://127.0.0.1:8000/')
catalog_url: http://127.0.0.1:8000/
```

After setting the controller URL (CATALOG_URL), the coordinator must specify the cryptographic backend and settings. observe that in our current set up, the Sphinx [3] decryption mix-net is the default option.

```
Select backend, one of SPHINXMIX, ZEUS (default: 'SPHINXMIX')
backend: SPHINXMIX
Set BODY_LEN (default: '1024')
BODY_LEN: 1024
Set GROUP (default: '713')
GROUP: 713
Set HEADER_LEN (default: '192')
HEADER_LEN: 192
```

`BODY_LEN, GROUP, HEADER_LEN` are settings related to Sphinx (e.g. `BODY_LEN` is the maximum length that a message may have). The coordinator is then instructed how to initialize the database and the server.

```
You need to setup your database once with
 panoramix-manage migrate
Start server with
 PANORAMIX_CONFIG=/tmp/panserver panoramix-manage runserver 127.0.0.1:8000
```

Then the coordinator can employ the contributor wizard to jointly set up the mix-net along with any other mix-net contributor. The coordinator first specifies the controller URL and selects the

cryptographic settings, which should match those of the controller.

```
% panoramix-wizard
Welcome to PANORAMIX wizard!
Configuration file is: /tmp/pancfg1
Set PANORAMIX_CONFIG environment variable to override
Set CATALOG_URL (default: 'http://127.0.0.1:8000/')
CATALOG_URL: http://127.0.0.1:8000/
Choose 'create' or 'join' mix-net
role: create
Select backend, one of SPHINXMIX, ZEUS (default: 'SPHINXMIX')
backend: SPHINXMIX
Set BODY_LEN (default: '1024')
BODY_LEN: 1024
Set GROUP (default: '713')
GROUP: 713
Set HEADER_LEN (default: '192')
HEADER_LEN: 192
```

Next, the coordinator creates and registers their personal cryptographic key to the controller.

```
No key available. Choose 'set' or 'create' (default: 'create')
action: create
Created key with values: {'SECRET':
  u'A808BCBBCC3BC141EA7B2FAFB5D', 'PUBLIC':
  '039073ae91caf2eed8971f26cfcb'}
Specify name to register as peer
PEER_NAME: peer1
Registered peer with PEER_ID: 039073ae91caf2eed8971f26cfcb
```

Then, the coordinator selects the mix-net attributes and initiates a negotiation in order to create the mix-net in agreement with other mix-net contributors. The coordinator invites them to join the process, by sharing with them the controller URL and a cryptographically secure invitation.

```
Choose mix-net peer name
name: our_mix-net
Give number of invitations to create
invitations: 1
Send invitations to peers:
  TLCWFVt8Y3gnvvtYNVRM0uM4|xHqbpCEmGirxpSsrByx9zQhm
Your initial proposal is contribution: 21
```

A.2 Contributor

Using the same wizard, the invited contributors can choose to approve the coordinator's proposal and finalize the creation of the mix-net. This is done via rounds of negotiations until all involved contributors agree upon the same proposal. The contributor will see the same messages but instead he or she will choose to *join* the mix-net.

```
% panoramix-wizard
Welcome to PANORAMIX wizard!
Configuration file is: /tmp/pancfg2
```

```
Set PANORAMIX_CONFIG environment variable to override
Set CATALOG_URL (default: 'http://127.0.0.1:8000/')
CATALOG_URL:
Choose 'create' or 'join' mix-net
role: join
Give invitation to create mix peer
JOIN_INVITATION:
TLCWFVt8Y3gnvvtYNVRM0uM4|xHqbpCEmGirxpSsrByx9zQhm
Negotiation initialized by peer
039073ae91caf2eed8971f26cfcb with contribution
21.
Proposed crypto backend: 'SPHINXMIX'; 'accept' or 'abort'? (default:
'accept')
response: accept
Proposed crypto params: '{u'BODY_LEN': 1024, u'GROUP': 713, u'HEADER_LEN':
  192}'; 'accept' or 'abort'? (default: 'accept')
response: accept
```

In the meantime, this message will be displayed to the coordinator.

```
Invitations pending: set(['xHqbpCEmGirxpSsrByx9z'])
All invited peers have joined. Sending accept contribution.
Your new contribution id is: 24
No consensus yet.
Consensus reached:
f1f3055b0be745c6224e9ad4f9512c9
Negotiation finished successfully. Applying consensus.
Created combined peer
0255e5a9676ad006a8443acf5fc45d.
```

Since the mix-net is now set up, we can give the end-users the mix-net URL, which can be used to access the mix-net.

```
http://127.0.0.1:8000/panoramix/peers/0255e5a9676ad006a8443acf5fc45d/
```

However, the mix-net is not ready to accept incoming messages, before all mix-net contributors agree to create and open an inbox. This is also be done through the wizard. The coordinator selects the inbox attributes and then it is up to the other contributors accept them through a round of negotiations.

```
Specify endpoint name to create on combined peer
ENDPOINT_NAME: gateway
Select endpoint type, one of SPHINXMIX_GATEWAY, SPHINXMIX (default:
SPHINXMIX_GATEWAY)
ENDPOINT_TYPE: SPHINXMIX_GATEWAY
Specify minimum size
MIN_SIZE: 3
Specify maximum size:
MAX_SIZE: 10
Give description:
EP_DESCRIPTION: the mix-net gateway
Contribution pending from: set([u'024f06abba6aa750acb07'])

...
```

```
All peer owners have agreed. Sending accept contribution.
Sent contribution 28
No consensus yet.
Consensus reached:
6e2ca8b1198efed79df24b988b6ce
Negotiation finished successfully. Applying consensus.
Created endpoint gateway_1.
```

Now the mix-net is ready to accept messages on the created inbox. The mix-net contributors can use the wizard to automate the processing of the inbox. The wizard polls the inbox to check whether it is ready to be processed, and facilitates the contributors to agree on the its closure.

```
Waiting until minimum inbox size is reached.
Contribution pending from:
set([u'024f06abba6aa750acb07'])
All peer owners have agreed. Sending accept contribution.
Sent contribution 34
Consensus reached:
365e84425bb914bb4342cb2df8e
Negotiation finished successfully. Applying consensus.
Closed endpoint gateway_1.
```

Next, each contributor retrieves the inbox messages, processes them locally using the cryptographic operation specified by the endpoint and uploads the processed messages. Depending on the application, messages may be forwarded to the contributors' own endpoints for further processing, until the final results reach the outbox.

```
Waiting to collect inbox.
Collected input for inbox of 039073a_for_ep_gateway_1.
Closed endpoint 039073a_for_ep_gateway_1
Processed endpoint 039073a_for_ep_gateway_1
Collected input for processbox of gateway_1.
Contribution pending from:
set([u'024f06abba6aa750acb07'])
All peer owners have agreed. Sending accept contribution.
Sent contribution 41
Consensus reached:
ddb5dfeb2adf62f682fb1500b2
Negotiation finished successfully. Applying consensus.
Processed endpoint gateway_1.
```

B. Documentation for Software Developers

B.1 Overview

As we discussed in D4.1, the PANORAMIX API is based around `peers` who exchange `messages`. Each peer performs supported operations (e.g. mixing, decrypting) through respective `endpoints`. Each endpoint processes messages in bulk: An endpoint `cycle` opens up in order to accept messages in its `inbox`. When sufficient messages are collected in the inbox, the peer retrieves the messages, processes them and posts them to its `outbox`. An external posting mechanism is responsible to send the outbox messages to the inboxes of their recipients.

B.2 Negotiations and Consensus

Negotiation is a mechanism that allows peers to agree upon a common text after rounds of amendments. The final text is signed by all participating peers. A text can be a prescription for an action that requires consensus of all stakeholders.

When a negotiation completes successfully, a consensus identifier is computed by hashing the negotiation data. This identifier can be provided to any operation that requires a consensus to proceed. For instance, in order to create a new peer with multiple owners there must be a consensus among the owners. The owners of a peer must also agree in order for any peer-related action to take place, for example to create an endpoint or to publish the endpoint's outbox.

B.2.1 Initiate a negotiation

The peer who starts a new negotiation is given a hard-to-guess negotiation id. The peer can then invite other peers to the negotiation by sharing the id with them.

URI	Method	Description
`/negotiations`	POST	Initiate a negotiation

Example request:

```
{
  "data": {},
  "info": {"resource": "negotiation", "operation": "create"},
  "meta": {"signature": "payload signature", "key_data": "public key"}
}
```

B.2.2 Get negotiation details

URI	Method	Description
/negotiations/<negotiation_id>	GET	Get negotiation details

Get negotiation details by id or consensus id.

Example response:

```
{
  "data": {
      "id": "long_negotiation_id",
      "text": null,
      "status": "OPEN",
      "timestamp": null,
      "consensus": null,
      "signings": [],
  }
}
```

Example response:

```
{
  "data": {
      "id": "long_negotiation_id",
      "text": "agreed upon text",
      "status": "DONE",
      "timestamp": "consensus timestamp",
      "consensus": "consensus hash",
      "signings": [{"signer_key_id": "peer1",
                    "signature": "peer1 sig"},
                   {"signer_key_id": "peer2",
                    "signature": "peer2 sig"}
                  ]
  }
}
```

B.2.3 Contribute to negotiation

Contribute a signed text to a negotiation. The text consists of the text body and a metadata dict. If all peers participating so far sign the same text that include the metadata `"accept": True`, then the negotiation completes successfully and the consensus id is produced. No more contributions are accepted.

Note: If the original contributor submits a text with `"accept": True`, the negotiation will complete successfully, although just one peer has contributed. Such a single-peer "consensus" may be useful in order to record a decision for an action in a uniform way regardless of the number of involved peers.

URI	Method	Description
/contributions/	POST	Contribute to negotiation

Example request:

```
{
  "data": {"negotiation_id": "neg_id",
           "text": "a text describing a peer creation",
```

```
            "signature": "text signature"},
  "info": {"resource": "contribution", "operation": "create"},
  "meta": {"signature": "payload signature", "key_data": "public key"}
}
```

Note: The contribution text should be a canonical representation of a dictionary of the following structure:

```
{
  "data": { "whatever data"},
  "info": { "whatever info" },
  "meta": {"accept": "true/false",
           "signers": [
             "signer_1", "signer_2", "signer_n"
          ]}
}
```

B.2.4 List contributions to a negotiation

URI	Method	Description
`/contributions/`	GET	List contributions to a negotiation

List contributions. Filtering by negotiation id is required.

Example response:

```
[{
  "data": {
    "id": "contribution_id", "negotiation_id": "neg_id",
    "text": "contribution text",
    "latest": true,
    "signer_key_id": "signer's public key",
    "signature": "signature",
  }
}]
```

B.3 Peers

A peer is any participant to the mix-net, either a mix-net contributor, a correspondent, an auditor, or any other stakeholder. A peer must be registered to the mix-net controller using a cryptographic identifier.

B.3.1 Create a Peer

Create a new peer with the specified parameters; see the example below. You must always provide a `consensus_id`, indicating a decision to create a peer agreed upon by all stakeholders through a negotiation. This applies for the simple case of creating a peer with no owners, as well.

URI	Method	Description
`/peers`	POST	Create a peer

Example request:

```
{
  "data": {"key_data": "public key",
           "key_id": "13C18335A029BEC5",
           "status": "READY",
           "owners": [{"owner_key_id": "owner1"},
                      {"owner_key_id": "owner2"}],
           "key_type": 1,
           "name": "peer1"},
  "info": {"operation": "create", "resource": "peer"},
  "by_consensus": {"consensus_id": "<consensus id>",
                   "consensus_type": "structural"},
  "meta": {"signature": "payload signature", "key_data": "public key"},
}
```

B.3.2 Get peer info

Get info for a single peer.

URI	Method	Description
/peers/<peer_id>	GET	Get info for a peer

Example response:

```
{
  "data": {"key_data": "public key",
           "key_id": "13C18335A029BEC5",
           "status": "READY",
           "name": "peer1",
           "key_type": 1,
           "key_type_params": "params",
           "owners": [{"owner_key_id": "owner1"},
                      {"owner_key_id": "owner2"}],
           "consensus_logs": [{"timestamp": "action timestamp",
                               "status": "READY",
                               "consensus_id": "consensus id"}]
         }
}
```

B.3.3 List Peers

Returns a list containing information about the registered peers.

URI	Method	Description
/peers	GET	List peers

Example response:

```
[{
   "data": { "some data..." }
}]
```

B.4 Endpoints

A peer handles messages in its endpoints. An endpoint specifies a type of operation along with relevant endpoint parameters, such as the minimum and maximum number of allowed messages. A correspondent sends messages to an open endpoint. Endpoint owners can agree to close the endpoint when suited and, after processing the inbox, publish the results in the outbox.

B.4.1 Create a peer endpoint

Creating an endpoint requires a consensus id, which proves the agreement of all peer owners on the action.

URI	Method	Description
`/endpoints`	POST	Create a peer endpoint

Example request:

```
{
  "data": {"peer_id": "13C18335A029BEC5",
           "endpoint_id": "identifier",
           "endpoint_type": "ZEUS_SK_MIX",
           "endpoint_params": "",
           "description": "a description",
           "status": "OPEN",
           "size_min": 10,
           "size_max": 1000},
  "info": {"operation": "create", "resource": "endpoint"},
  "by_consensus": {"consensus_id": "<consensus id>",
                   "consensus_type": "structural"},
  "meta": {"signature": "payload signature", "key_data": "public key"},
}
```

B.4.2 Update an endpoint

The status of an endpoint can be updated, given the last consensus id and status-specific required data.

URI	Method	Description
`/endpoint/<endpoint_id>`	PATCH	Partially update an endpoint

Example request:

```
{
  "data": {"endpoint_id": "identifier",
           "status": "PROCESSED",
           "message_hashes": ["a processed message hash"],
           "process_proof": "the processing proof",
          },
  "info": {"operation": "partial_update",
           "resource": "endpoint",
           "on_last_consensus_id": "previous consensus"},
  "meta": {"signature": "payload signature", "key_data": "public key"},
}
```

B.4.3 Get endpoint info

URI	Method	Description
`/endpoint/<endpoint_id>`	GET	Get info for an endpoint

Example response:

```
{
  "data": {"peer_id": "13C18335A029BEC5",
           "endpoint_id": "identifier",
           "endpoint_type": "ZEUS_SK_MIX",
           "endpoint_params": "",
           "description": "a description",
           "status": "CLOSED",
           "size_min": 10,
           "size_max": 1000,
           "inbox_hash": "inbox hash",
           "last_message_id": "message_id",
           "consensus_logs": [{"timestamp": "open action timestamp",
                               "status": "OPEN",
                               "consensus_id": "consensus id1"},
                              {"timestamp": "close action timestamp",
                               "status": "CLOSED",
                               "consensus_id": "consensus id2"}]
         }
}
```

B.4.4 List endpoints

URI	Method	Description
`/endpoints`	GET	List endpoints

Example response:

```
[{
   "data": { "some data..." }
}]
```

B.5 Messages

Messages are posted to an endpoint's `inbox` of a specified peer. Once a sufficient number of messages are collected, the peer retrieves the inbox messages, processes them and uploads the transformed messages to the `processbox`. Once the peer owners agree on the results and mark the endpoint as PROCESSED (see above), the processed messages move to the `outbox`.

B.5.1 Send a message to inbox/processbox

URI	Method	Description
`/messages`	POST	Send a message

No consensus is needed in order to send a message.

Example request:

```
{
  "data": {"endpoint_id": "endpoint name",
           "box": "INBOX",
           "sender": "FC650CA0F7749FF0",
           "recipient": "13C18335A029BEC5",
           "text": "encrypted message"
          },
  "info": {"operation": "create", "resource": "message"},
  "meta": {"signature": "payload signature", "key_data": "public key"}
}
```

B.5.2 List messages

One can list the messages of a specified endpoint and box.

URI	Method	Description
`/messages`	GET	List messages

Example inbox response:

```
[
  {"data": {"endpoint_id": "endpoint name",
            "box": "INBOX",
            "id": 1,
            "sender": "orig_sender1",
            "recipient": "this_peer",
            "text": "encrypted message 1",
            "message_hash": "msg hash 1"}
  },
  {"data": {"endpoint_id": "endpoint name",
            "box": "INBOX",
            "id": 2,
            "sender": "orig_sender2",
            "recipient": "this_peer",
            "text": "encrypted message 2",
            "message_hash": "msg hash 2"}
  }
]
```

Example outbox response:

```
[
  {"data": {"endpoint_id": "endpoint name",
            "box": "OUTBOX",
            "id": 3,
            "sender": "this_peer",
            "recipient": "next_peer_a",
            "text": "decrypted message a",
            "message_hash": "msg hash a"}
  },
  {"data": {"endpoint_id": "endpoint name",
            "box": "OUTBOX",
            "id": 4,
            "sender": "this_peer",
            "recipient": "next_peer_b",
            "text": "decrypted message b",
            "message_hash": "msg hash b"}
```

```
  }
]
```

In this example, we assume that processing has shuffled the messages in order to hide the connection between encrypted messages (1, 2) and decrypted messages (a and b).

Bibliography

[1] The European Union general data protection regulation. `http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf`, 2016. [Online; accessed 30-November-2017].

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

[3] 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.

[4] 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.