Panoramix Documentation

Release 0.1.dev0

GRNET

Jun 02, 2019

CONTENTS

Contents:

CHAPTER

ONE

PANORAMIX API

This is a guide for the Panoramix API, which provides functionality to create and operate mixnets.

1.1 Overview

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 their recipients.

1.2 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.

1.2.1 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"}
}
```

1.2.2 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": None,
    "status": "OPEN",
    "timestamp": None,
    "consensus": None,
    "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"}
                ]
  }
}
```

1.2.3 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": {...},
  "info": {...},
  "meta": {"accept": bool,
           "signers": list,
           ...}
}
```

1.2.4 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",
   }
}]
```

1.3 Peers

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

1.3.1 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"},
}
```

1.3.2 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"}]
          }
}
```

1.3.3 List Peers

Returns a list containing information about the registered peers.

URI	Method	Description
/peers	GET	List peers

Example response:

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

1.4 Endpoints

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

1.4.1 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"},
}
```

1.4.2 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",
          }
```

(continues on next page)

(continued from previous page)

```
  "info": {"operation": "partial_update",
           "resource": "endpoint",
           "on_last_consensus_id": "previous consensus"},
  "meta": {"signature": "payload signature", "key_data": "public key"},
}
```

1.4.3 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"}]
          }
}
```

1.4.4 List endpoints

URI	Method	Description
/endpoints	GET	List endpoints

Example response:

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

1.5 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*.

1.5.1 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"}
}
```

1.5.2 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).

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search